Table of Contents

What is Lumberyard? .......................................................................................................................... 1
  Professional-Grade AAA Engine ........................................................................................................ 1
      Beautiful Worlds .......................................................................................................................... 2
  Asset Changes ................................................................................................................................. 2
  Component Entities .......................................................................................................................... 2
  Compelling Characters .................................................................................................................... 2
  Robust Networking .......................................................................................................................... 2
  Real-Time Gameplay Editing ........................................................................................................... 3
  Modular Gems .................................................................................................................................. 3
  Wwise LTX ...................................................................................................................................... 3
Integrated with AWS ......................................................................................................................... 3
  Amazon GameLift ............................................................................................................................ 3
  Cloud Canvas .................................................................................................................................. 3
  AWS SDK for C++ ............................................................................................................................ 3
Integrated with Twitch ....................................................................................................................... 4
  Twitch ChatPlay .............................................................................................................................. 4
Free with Source ............................................................................................................................... 4
Lumberyard Systems .......................................................................................................................... 4
Supported platforms .......................................................................................................................... 7
Set up .................................................................................................................................................. 7
      System requirements ...................................................................................................................... 8
      Install .......................................................................................................................................... 10
      Upgrade ....................................................................................................................................... 12
      Set Up a Development Environment .......................................................................................... 16
      Enable a Firewall ......................................................................................................................... 23
      Set Up Perforce ............................................................................................................................ 23
      Uninstall ..................................................................................................................................... 24
Programming Concepts ..................................................................................................................... 25
      Asset System ............................................................................................................................... 25
      AZ Code Generator ....................................................................................................................... 25
      Building ....................................................................................................................................... 26
      Cloud Canvas ............................................................................................................................... 26
      Debugging ................................................................................................................................... 26
      EBuses .......................................................................................................................................... 26
      Entities and Components .............................................................................................................. 26
      Gems and AZ Modules ................................................................................................................... 27
      Input ............................................................................................................................................... 27
      Lua ................................................................................................................................................ 28
      Networking ................................................................................................................................... 28
      Reflection System ........................................................................................................................ 28
      Script Canvas ............................................................................................................................... 28
      Slices .......................................................................................................................................... 28
      C++ Best Practices ....................................................................................................................... 28
Editors and Tools ................................................................................................................................. 32
  Asset File Types ............................................................................................................................... 35
Blog, Forums, and Feedback ............................................................................................................... 37
Key Terminology ................................................................................................................................. 37
Lumberyard projects ............................................................................................................................ 43
  Working with projects in Lumberyard ............................................................................................... 43
Create a project ................................................................................................................................. 43
    Creating and Switching Game Projects .......................................................................................... 44
    Configuring Advanced Settings .................................................................................................. 49
    Understanding the Project Configurator ..................................................................................... 53
Troubleshooting ................................................................................................................................. 55

Version 1.28

iii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Outliner</td>
<td>462</td>
</tr>
<tr>
<td>Working with Entities</td>
<td>463</td>
</tr>
<tr>
<td>Working with Layers</td>
<td>469</td>
</tr>
<tr>
<td>Entity Inspector</td>
<td>475</td>
</tr>
<tr>
<td>Setting an Entity Status</td>
<td>476</td>
</tr>
<tr>
<td>Customizing the Entity Icon</td>
<td>477</td>
</tr>
<tr>
<td>Pinning an Entity's Inspector</td>
<td>478</td>
</tr>
<tr>
<td>Working with Components</td>
<td>479</td>
</tr>
<tr>
<td>Adding Components to an Entity</td>
<td>479</td>
</tr>
<tr>
<td>Managing Components</td>
<td>480</td>
</tr>
<tr>
<td>Editing Component Properties</td>
<td>481</td>
</tr>
<tr>
<td>Disabling and Enabling Components on Entities</td>
<td>482</td>
</tr>
<tr>
<td>Editing Components in the Viewport</td>
<td>483</td>
</tr>
<tr>
<td>Using Manipulators</td>
<td>490</td>
</tr>
<tr>
<td>Working with the Viewport Interaction Model</td>
<td>494</td>
</tr>
<tr>
<td>Working with Slices</td>
<td>510</td>
</tr>
<tr>
<td>Anatomy of a Slice</td>
<td>514</td>
</tr>
<tr>
<td>Creating a Slice</td>
<td>516</td>
</tr>
<tr>
<td>Instantiating a Slice</td>
<td>517</td>
</tr>
<tr>
<td>Creating Nested Slices</td>
<td>517</td>
</tr>
<tr>
<td>Modifying a Slice and Saving Changes</td>
<td>519</td>
</tr>
<tr>
<td>Detaching Entities and Slice Instances</td>
<td>522</td>
</tr>
<tr>
<td>Reverting and Forcing Overrides</td>
<td>522</td>
</tr>
<tr>
<td>Slice Reloading</td>
<td>524</td>
</tr>
<tr>
<td>Working with Dynamic Slices</td>
<td>524</td>
</tr>
<tr>
<td>Slice Favorites</td>
<td>526</td>
</tr>
<tr>
<td>Slice Upgrade Pipeline</td>
<td>529</td>
</tr>
<tr>
<td>Component Reference</td>
<td>532</td>
</tr>
<tr>
<td>AI</td>
<td>532</td>
</tr>
<tr>
<td>Animation</td>
<td>533</td>
</tr>
<tr>
<td>Audio</td>
<td>533</td>
</tr>
<tr>
<td>Camera</td>
<td>533</td>
</tr>
<tr>
<td>Editor</td>
<td>533</td>
</tr>
<tr>
<td>Environment</td>
<td>533</td>
</tr>
<tr>
<td>Gameplay</td>
<td>534</td>
</tr>
<tr>
<td>Network</td>
<td>534</td>
</tr>
<tr>
<td>PhysX</td>
<td>534</td>
</tr>
<tr>
<td>Physics (Legacy)</td>
<td>534</td>
</tr>
<tr>
<td>Rendering</td>
<td>534</td>
</tr>
<tr>
<td>Scripting</td>
<td>535</td>
</tr>
<tr>
<td>Shape</td>
<td>535</td>
</tr>
<tr>
<td>Terrain</td>
<td>535</td>
</tr>
<tr>
<td>UI</td>
<td>535</td>
</tr>
<tr>
<td>VR</td>
<td>535</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>536</td>
</tr>
<tr>
<td>Actor</td>
<td>536</td>
</tr>
<tr>
<td>AnimGraph</td>
<td>545</td>
</tr>
<tr>
<td>Anim Graph Net Sync</td>
<td>545</td>
</tr>
<tr>
<td>Area Light</td>
<td>546</td>
</tr>
<tr>
<td>Attachment</td>
<td>550</td>
</tr>
<tr>
<td>Audio Area Environment</td>
<td>551</td>
</tr>
<tr>
<td>Audio Environment</td>
<td>552</td>
</tr>
<tr>
<td>Audio Listener</td>
<td>553</td>
</tr>
<tr>
<td>Audio Preload</td>
<td>555</td>
</tr>
<tr>
<td>Audio Proxy</td>
<td>557</td>
</tr>
<tr>
<td>Audio Rtpc</td>
<td>557</td>
</tr>
<tr>
<td>Audio Switch</td>
<td>558</td>
</tr>
<tr>
<td>Component/Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>PhysX Shape Collider</td>
<td>763</td>
</tr>
<tr>
<td>PhysX Terrain</td>
<td>767</td>
</tr>
<tr>
<td>Rain</td>
<td>768</td>
</tr>
<tr>
<td>Random Timed Spawner</td>
<td>776</td>
</tr>
<tr>
<td>Render to Texture</td>
<td>781</td>
</tr>
<tr>
<td>River</td>
<td>796</td>
</tr>
<tr>
<td>Road</td>
<td>808</td>
</tr>
<tr>
<td>Script Canvas</td>
<td>818</td>
</tr>
<tr>
<td>Shapes: Cylinder, Capsule, Disk, Box, Sphere, Compound</td>
<td>818</td>
</tr>
<tr>
<td>Simple Motion</td>
<td>826</td>
</tr>
<tr>
<td>Simple State</td>
<td>828</td>
</tr>
<tr>
<td>Sky Cloud</td>
<td>830</td>
</tr>
<tr>
<td>Sky Highlight</td>
<td>847</td>
</tr>
<tr>
<td>Snow</td>
<td>853</td>
</tr>
<tr>
<td>Spawner</td>
<td>859</td>
</tr>
<tr>
<td>Spline</td>
<td>866</td>
</tr>
<tr>
<td>Tag</td>
<td>875</td>
</tr>
<tr>
<td>Transform</td>
<td>878</td>
</tr>
<tr>
<td>Trigger Area</td>
<td>890</td>
</tr>
<tr>
<td>Tube Shape</td>
<td>894</td>
</tr>
<tr>
<td>UI Canvas Asset Ref</td>
<td>897</td>
</tr>
<tr>
<td>UI Canvas on Mesh</td>
<td>897</td>
</tr>
<tr>
<td>UI Canvas Proxy Ref</td>
<td>898</td>
</tr>
<tr>
<td>Vegetation Layer Spawner</td>
<td>900</td>
</tr>
<tr>
<td>Vertex Containers</td>
<td>904</td>
</tr>
<tr>
<td>Video Playback</td>
<td>906</td>
</tr>
<tr>
<td>Video Playback Bink</td>
<td>913</td>
</tr>
<tr>
<td>VisArea</td>
<td>918</td>
</tr>
<tr>
<td>VR Preview</td>
<td>924</td>
</tr>
<tr>
<td>Water Volume</td>
<td>925</td>
</tr>
<tr>
<td>White Box</td>
<td>946</td>
</tr>
<tr>
<td>White Box Collider</td>
<td>960</td>
</tr>
<tr>
<td>Converting Entities with the Legacy Converter</td>
<td>963</td>
</tr>
<tr>
<td>Converting Your Legacy Entities</td>
<td>963</td>
</tr>
<tr>
<td>List of Legacy Entity Conversions</td>
<td>964</td>
</tr>
<tr>
<td>Programmer's Guide to Entities and Components</td>
<td>973</td>
</tr>
<tr>
<td>Creating a Component</td>
<td>974</td>
</tr>
<tr>
<td>Registering Your Component</td>
<td>977</td>
</tr>
<tr>
<td>Reflecting a Component for Serialization and Editing</td>
<td>977</td>
</tr>
<tr>
<td>Defining and Using Component Services</td>
<td>983</td>
</tr>
<tr>
<td>Editor Components</td>
<td>984</td>
</tr>
<tr>
<td>Creating System Components</td>
<td>987</td>
</tr>
<tr>
<td>Programmer's Guide to Component Mode</td>
<td>992</td>
</tr>
<tr>
<td>Components and EBuses</td>
<td>1015</td>
</tr>
<tr>
<td>Tick Bus and Components</td>
<td>1019</td>
</tr>
<tr>
<td>Exposing Custom Components to Track View for Animation</td>
<td>1021</td>
</tr>
<tr>
<td>Components and EBuses: Best Practices</td>
<td>1026</td>
</tr>
<tr>
<td>Gameplay Bus</td>
<td>1030</td>
</tr>
<tr>
<td>GameplayNotificationId</td>
<td>1030</td>
</tr>
<tr>
<td>GameplayNotifications</td>
<td>1031</td>
</tr>
<tr>
<td>Script Examples</td>
<td>1031</td>
</tr>
<tr>
<td>Reflecting Lumberyard Classes, Methods, and EBus Interfaces</td>
<td>1032</td>
</tr>
<tr>
<td>Serialization Context</td>
<td>1033</td>
</tr>
<tr>
<td>Edit Context</td>
<td>1052</td>
</tr>
<tr>
<td>Behavior Context</td>
<td>1053</td>
</tr>
<tr>
<td>Gems</td>
<td>1064</td>
</tr>
<tr>
<td>Enabling Gems</td>
<td>1064</td>
</tr>
<tr>
<td>Animation</td>
<td>1340</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Cinematics</td>
<td>1583</td>
</tr>
<tr>
<td>Particle effects</td>
<td>1504</td>
</tr>
<tr>
<td>Particle Debugging with Console Variables</td>
<td>1578</td>
</tr>
<tr>
<td>Advanced Particle Techniques</td>
<td>1576</td>
</tr>
<tr>
<td>Particle Effects Best Practices</td>
<td>1576</td>
</tr>
<tr>
<td>Using the Particle Editor</td>
<td>1504</td>
</tr>
<tr>
<td>Best Practices for Working with Skinned Meshes for Animation</td>
<td>1492</td>
</tr>
<tr>
<td>Level creation</td>
<td>1219</td>
</tr>
<tr>
<td>Levels and environment effects</td>
<td>1219</td>
</tr>
<tr>
<td>Creating a New Level</td>
<td>1219</td>
</tr>
<tr>
<td>Creating Terrain</td>
<td>1220</td>
</tr>
<tr>
<td>Blending Environment Probes</td>
<td>1252</td>
</tr>
<tr>
<td>Working with Database Libraries</td>
<td>1264</td>
</tr>
<tr>
<td>Adding Sky Effects</td>
<td>1265</td>
</tr>
<tr>
<td>Adding Weather Effects</td>
<td>1290</td>
</tr>
<tr>
<td>Adding Vegetation</td>
<td>1298</td>
</tr>
<tr>
<td>Dynamic vegetation</td>
<td>1314</td>
</tr>
<tr>
<td>Dynamic Vegetation Concepts</td>
<td>1316</td>
</tr>
<tr>
<td>Dynamic Vegetation Procedures</td>
<td>1319</td>
</tr>
<tr>
<td>Animation</td>
<td>1340</td>
</tr>
<tr>
<td>Working with animation in Lumberyard</td>
<td>1340</td>
</tr>
<tr>
<td>Create and animate characters</td>
<td>1340</td>
</tr>
<tr>
<td>Animation Editor Concepts and Terms</td>
<td>1341</td>
</tr>
<tr>
<td>Animation Editor User Interface</td>
<td>1348</td>
</tr>
<tr>
<td>Animation Editor File Types</td>
<td>1352</td>
</tr>
<tr>
<td>Getting Started with the Animation Editor</td>
<td>1353</td>
</tr>
<tr>
<td>Referencing External Anim Graphs</td>
<td>1360</td>
</tr>
<tr>
<td>Synchronizing Animation Graphs</td>
<td>1372</td>
</tr>
<tr>
<td>Animation Editor Components</td>
<td>1398</td>
</tr>
<tr>
<td>Using Morph Targets to Deform Characters</td>
<td>1399</td>
</tr>
<tr>
<td>Customizing State Machine Routing with Sparse Motion Sets</td>
<td>1413</td>
</tr>
<tr>
<td>Animation Editor Nodes</td>
<td>1415</td>
</tr>
<tr>
<td>Using Tags with Animation Graphs</td>
<td>1446</td>
</tr>
<tr>
<td>Customizing EMotion FX Objects</td>
<td>1449</td>
</tr>
<tr>
<td>Creating Custom Motion Events and Parameters Using C++</td>
<td>1451</td>
</tr>
<tr>
<td>Creating and Simulating a PhysX Ragdoll</td>
<td>1458</td>
</tr>
<tr>
<td>Creating Additive Animations</td>
<td>1470</td>
</tr>
<tr>
<td>Retargeting Motions</td>
<td>1471</td>
</tr>
<tr>
<td>Creating Simulated Objects</td>
<td>1473</td>
</tr>
<tr>
<td>Add Cloth Colliders to actors</td>
<td>1489</td>
</tr>
<tr>
<td>Best Practices for Working with Skinned Meshes for Animation</td>
<td>1492</td>
</tr>
<tr>
<td>Particle effects</td>
<td>1504</td>
</tr>
<tr>
<td>Using the Particle Editor</td>
<td>1504</td>
</tr>
<tr>
<td>Particle Effects Best Practices</td>
<td>1576</td>
</tr>
<tr>
<td>Advanced Particle Techniques</td>
<td>1576</td>
</tr>
<tr>
<td>Particle Debugging with Console Variables</td>
<td>1578</td>
</tr>
<tr>
<td>Cinematics</td>
<td>1583</td>
</tr>
<tr>
<td>Using the Track View Editor</td>
<td>1584</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Overview</td>
<td>2108</td>
</tr>
<tr>
<td>Cloud Gems</td>
<td>2127</td>
</tr>
<tr>
<td>Gameplay Design and Engineering Guide</td>
<td>2221</td>
</tr>
<tr>
<td>Software Engineering Guide</td>
<td>2224</td>
</tr>
<tr>
<td>Administering Cloud Canvas</td>
<td>2348</td>
</tr>
<tr>
<td>Using the Command Line</td>
<td>2390</td>
</tr>
<tr>
<td>Localization</td>
<td>2415</td>
</tr>
<tr>
<td>Configuring and Loading Localization Data</td>
<td>2415</td>
</tr>
<tr>
<td>Localization Checklist</td>
<td>2416</td>
</tr>
<tr>
<td>UI System Localization Features</td>
<td>2416</td>
</tr>
<tr>
<td>Font Features</td>
<td>2417</td>
</tr>
<tr>
<td>Localization Initialization</td>
<td>2417</td>
</tr>
<tr>
<td>Scripting</td>
<td>2420</td>
</tr>
<tr>
<td>Script Canvas</td>
<td>2420</td>
</tr>
<tr>
<td>Learning Guide</td>
<td>2421</td>
</tr>
<tr>
<td>Common Tasks</td>
<td>2468</td>
</tr>
<tr>
<td>Editor Reference</td>
<td>2479</td>
</tr>
<tr>
<td>Programmer Guide</td>
<td>2543</td>
</tr>
<tr>
<td>Script Canvas Node Reference</td>
<td>2567</td>
</tr>
<tr>
<td>Lua Scripting</td>
<td>2683</td>
</tr>
<tr>
<td>Learning Lua</td>
<td>2684</td>
</tr>
<tr>
<td>Learning Lua in Lumberyard</td>
<td>2684</td>
</tr>
<tr>
<td>Adding Lua Scripts to Component Entities</td>
<td>2684</td>
</tr>
<tr>
<td>Basic Structure of a Component Entity Lua Script</td>
<td>2686</td>
</tr>
<tr>
<td>Properties Table</td>
<td>2687</td>
</tr>
<tr>
<td>Network Binding Properties</td>
<td>2691</td>
</tr>
<tr>
<td>Using EBus in Lua</td>
<td>2693</td>
</tr>
<tr>
<td>Lua Editor</td>
<td>2696</td>
</tr>
<tr>
<td>Debugging Lua Scripts</td>
<td>2703</td>
</tr>
<tr>
<td>The Lua Environment (Advanced)</td>
<td>2705</td>
</tr>
<tr>
<td>Component Entity Lua API Reference</td>
<td>2705</td>
</tr>
<tr>
<td>Script Events</td>
<td>2762</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>2762</td>
</tr>
<tr>
<td>Creating Script Events in Script Canvas</td>
<td>2762</td>
</tr>
<tr>
<td>Using Script Events in Script Canvas</td>
<td>2765</td>
</tr>
<tr>
<td>Using Script Events in Lua</td>
<td>2766</td>
</tr>
<tr>
<td>Script Events Best Practices</td>
<td>2769</td>
</tr>
<tr>
<td>PhysX system</td>
<td>2772</td>
</tr>
<tr>
<td>PhysX Gems</td>
<td>2772</td>
</tr>
<tr>
<td>PhysX Components</td>
<td>2773</td>
</tr>
<tr>
<td>PhysX Configuration</td>
<td>2773</td>
</tr>
<tr>
<td>PhysX Materials</td>
<td>2773</td>
</tr>
<tr>
<td>PhysX Debugging</td>
<td>2774</td>
</tr>
<tr>
<td>Configuring the PhysX System</td>
<td>2774</td>
</tr>
<tr>
<td>Global Configuration</td>
<td>2775</td>
</tr>
<tr>
<td>Collision Layers</td>
<td>2785</td>
</tr>
<tr>
<td>Collision Groups</td>
<td>2787</td>
</tr>
<tr>
<td>Debugger Configuration</td>
<td>2790</td>
</tr>
<tr>
<td>PhysX World Programming Notes</td>
<td>2796</td>
</tr>
<tr>
<td>Physics materials</td>
<td>2797</td>
</tr>
<tr>
<td>Physics material properties</td>
<td>2798</td>
</tr>
<tr>
<td>Create a physics material</td>
<td>2799</td>
</tr>
<tr>
<td>Assign a physics material library</td>
<td>2800</td>
</tr>
<tr>
<td>Assign a physics material</td>
<td>2802</td>
</tr>
<tr>
<td>Assign physics materials per face</td>
<td>2802</td>
</tr>
<tr>
<td><strong>Terrain physics materials</strong></td>
<td>2803</td>
</tr>
<tr>
<td>PhysX Scene Queries</td>
<td>2804</td>
</tr>
</tbody>
</table>
What is Lumberyard?

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Amazon Lumberyard is a free, cross-platform, 3D game engine that allows you to create high-quality games, connect your games to the compute and storage of the AWS Cloud, and engage fans on Twitch. With Lumberyard, you can spend more time creating great gameplay and building communities of fans, and less time on the heavy lifting of building a game engine and managing servers.

Lumberyard offers everything a professional game developer can expect, such as a full-featured editor, native code performance, and stunning visuals. It also includes hundreds of other ready-to-use features like networking, cinematics, the Script Canvas editor, the Animation Editor, audio tools, and more.

Interested? Want to get started?

- Download the latest version of the Lumberyard beta
- Kickstart your learning by watching tutorial videos
- Sign up for and participate in the Amazon Lumberyard forums

Image from Starter Game. For more sample projects and levels, see Using Lumberyard sample projects and levels (p. 142).

Professional-Grade AAA Engine

Lumberyard helps you build rich, engaging, world-class games through its comprehensive and proven toolset and highly optimized runtime performance.
Beautiful Worlds

The visuals technology of Lumberyard can bring to life any virtual environment. Your artists get a powerful toolbox to create high quality visual elements, such as physically based shaders, dynamic global illumination, particle effects, lifelike vegetation, real-time dynamic water caustics, and volumetric fog. They can also produce cinematic features like color grading, motion blur, depth of field, and integrated HDR lens flares.

For more information, see the following topics:

- Creating levels and adding environment effects (p. 1219)
- Adding particle effects (p. 1504)
- Create screen effects (p. 1811)

Asset Changes

With the Lumberyard's Asset Processor, you can quickly get assets into the engine. Save a file (for example, from Maya or Photoshop) into a directory, and the Asset Processor automatically processes that file from source art into game-ready assets. If you edit an asset, Lumberyard detects the change and automatically updates it in the background.

For more information, see Using Asset Processor (p. 248).

Component Entities

The component entity system helps you compose complex entities out of simpler entities. Content creators can drag and drop components to build behaviors, edit component settings live in the editor, and create Lua scripts to quickly change or extend an entity's behavior. To build and iterate for complex scenes, entities can be grouped together into slices. Those slices can in turn be used to create more complex slices, resulting in a fully cascading prefab system. If you change a slice, you can share your changes to all the slice instances, or keep your changes exclusive to just that instance.

For more information, see the following topics:

- Working with component entities (p. 462)
- Working with Slices (p. 510)

Compelling Characters

Artists can use Lumberyard to create believable characters and high-fidelity performances. Lumberyard's character tool, Animation Editor, combines animation, attachments, and physics simulations with blend shape, blend space, and animation layering.

For more information, see Create and animate characters (p. 1340).

Robust Networking

Lumberyard introduces GridMate, a robust and flexible networking solution designed for efficient bandwidth usage and low-latency communications. You can synchronize objects over the network with GridMate's replica framework. GridMate's session management can be integrated with major online console services and helps you handle peer-to-peer and client–server topologies with host migration.

For more information, see Using Lumberyard Networking (p. 1990).
Real-Time Gameplay Editing

Real-time gameplay editing helps you iterate on gameplay and immediately see your results without waiting for builds or leaving the editor. Lumberyard's Asset Processor automatically converts and optimizes your game assets in real time so that you can import game objects, fine-tune behavior, and play the game that you created.

Modular Gems

Lumberyard's Modular Gems system gives you a library of prebuilt features with which you can start new projects or prototype ideas quickly. Modular Gems offer increased control over which technologies to include in your game project. Create your own modular Gems or use the Gems included with Lumberyard, such as physics, multiplayer, "white box" rapid prototyping tools, and more.

For more information, see Add modular features and assets with Gems (p. 1064).

Wwise LTX

Lumberyard includes a version of Audiokinetic's advanced, feature-rich sound engine. Sound designers and composers can author rich soundscapes for your games.

For more information, see Adding Audio and Sound Effects (p. 3140).

Integrated with AWS

Lumberyard is deeply integrated with AWS so you can build live and multiplayer games with dramatically less cost, time, and technical risk. AWS integrations include:

Amazon GameLift

Amazon GameLift is an AWS service for deploying, operating, and scaling session-based multiplayer games. You can scale high-performance game servers up and down to meet player demand without additional engineering effort.

For more information, see the Amazon GameLift Developer Guide.

Cloud Canvas

Cloud Canvas is Lumberyard's technology for connecting your game to Amazon Web Services. With Cloud Canvas, you can use AWS to implement cloud-hosted features and create asynchronous multiplayer games. Using cloud gems, you can build gameplay that connects to AWS services, such as Amazon DynamoDB, AWS Lambda, and Amazon S3.

For more information, see Implementing Connected Features with Cloud Canvas (p. 2106).

AWS SDK for C++

The AWS SDK for C++ provides C++ API operations for numerous AWS services including Amazon S3, Amazon EC2, Amazon DynamoDB, and more, with support for all major native platforms. You can use the SDK to integrate AWS components into your game.
For more information, see the AWS SDK for C++.

Integrated with Twitch

Lumberyard is integrated with Twitch so that you can build games that engage with more than 1.7 million monthly broadcasters and more than 100 million monthly viewers on Twitch.

Twitch ChatPlay

The Twitch ChatPlay feature within Lumberyard helps you build gameplay that interacts in real time with Twitch viewers. For example, you can build a game where viewers can vote on game outcomes, gift power-ups to their favorite players, or change the level based on the number of viewers watching the player.

For more information, see Twitch ChatPlay System (p. 2096).

Free with Source

Lumberyard is free, including source code. You can customize Lumberyard for your team and vision for your project today, and for future projects. There are no seat fees, subscription fees, or requirements to share revenue. Only pay for the AWS services that you choose to use.

For more information, see the Lumberyard Licensing FAQ.

Lumberyard Systems

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard consists of the following major systems that help you develop levels within the editor:

- Adding Audio and Sound Effects (p. 3140)
  
  The audio translation layer (ATL) provides an interface between Lumberyard and third-party audio middleware so you can change your audio implementation without affecting the game logic.

- Create and animate characters (p. 1340)
  
  The character animation system combines skeletal-based deformation of meshes with morph-based vertex deformation to facilitate complex animation. You can create realistic character movements by playing and blending animation sequences, controlling facial expressions, and applying damage effects. Characters can play scripted movements, employ AI navigation.

- Create cinematic sequences (p. 1583)
  
  Cinematics are interactive movie animations. You can use Lumberyard to add cutscenes to your game. You can also add scripted events so that a sequence of objects, animations, and sounds are triggered in the game. The players can view these from their own (first person) or another's (third person) perspective.
• **Working with component entities (p. 462)**

The component entity system provides a modular and intuitive construction of game elements, such as lights, cameras, trigger areas, and objects. The component entity system works at both the system level and the entity level. It employs reflection, serialization, event bus (EBus) messaging, fully cascading slices, and the ability to drag and drop and edit entities and their components in Lumberyard Editor.

• **Add modular features and assets with Gems (p. 1064)**

Gems are packages that contain code and assets to augment your game projects, such as the ChatPlay and Woodland Asset Collection gems. You can select gems to include in your project through the Lumberyard Project Configurator (p. 43) and from the command line.

• **Creating levels and adding environment effects (p. 1219)**

A level, also known as world or map, represents the space or area available to the player while completing a game objective. A level's environment includes lighting, terrain, bodies of water, vegetation, sky, and weather effects.

• **Materials (p. 1636)**

You can use the **Material Editor** to create and apply materials, map textures, set opacity and lighting effects, set shader parameters, create vertex deformations, tessellation, and more.

• **Adding particle effects (p. 1504)**

The particle effects system simulates explosions, fire, smoke, sparks, water spray, fog, snow, rain, and other effects.

• **Script Canvas (p. 2420)**

Create game logic and behaviors with Lumberyard's new visual scripting environment.

• **Shader Rendering System (p. 1639)**

Lumberyard's physically based rendering (PBR) shaders use real-world physical rules and properties to describe how incoming light interacts with objects. This means that object materials look more convincing under different lighting conditions.

• **Twitch ChatPlay System (p. 2096)**

The Twitch ChatPlay Gem provides a flexible framework to create customized game interactions between broadcasters and spectators on Twitch, the world's leading social video platform and community for gamers. Twitch ChatPlay includes support for chat commands, polls, and surveys that can be triggered by Twitch viewers through the Twitch chat channel.
• **Creating and Customizing Project User Interfaces (p. 2874)**

With the **UI Editor**, you can create and customize various parts of the game user interface, such as images, text, buttons, menus, scroll boxes, and heads-up displays (HUDs).

• **Create virtual reality projects in Lumberyard (p. 3248)**

Virtual reality (VR) is a technology that replicates the gaming environment and simulates a user's presence in it. With virtual reality players, feel as if they are in the game world as they interact with the environment, characters, and objects. Lumberyard's virtual reality system integrates the use of the Oculus Rift and HTC Vive head-mounted displays (HMD) on PC gaming systems.

The following Lumberyard tools are outside of the main editor and are used for project and game development:

• **Working with the Asset Pipeline and asset files (p. 247)**

The Asset Pipeline converts source art and other assets into platform-specific, game ready data.

• **Managing Game Projects with Lmbr.exe (p. 56)**

Lmbr.exe is a command-line tool for managing capabilities, game projects, and gems.

• **Developing for Android and iOS with Lumberyard (p. 3166)**

You can use Lumberyard to build your games for Android devices such as the NVIDIA Shield, Samsung Galaxy Note 5, and Motorola Nexus 6, and iOS devices that use the A8 GPUs or later. See Apple's GPU list for a list of these devices.

• **Create macOS projects in Lumberyard (p. 3263)**

You can use Lumberyard to build macOS applications.

• **Creating Lumberyard Executables for Linux (p. 3269)**

Lumberyard supports compiling and deploying the Windows client for a multiplayer project on a Linux dedicated server.

• **Project Configurator (p. 43)**

The Project Configurator is a standalone application that allows you to specify to the Waf build system which game projects and assets (gems) to include in a game build. With it, you can create new projects, save active projects, configure advanced settings, and enable, disable, or create new gems.

• **Profiling, Testing, and Debugging Game Projects (p. 1920)**

Lumberyard includes a number of tools for testing builds, profiling performance, and debugging.
Using the Waf Build System (p. 63)

With the Waf build system, you can switch between various build pipelines and ensure that you build only what is needed. You can use extensions, such as automatic project generation, or a simple GUI to modify the command line base system for your project requirements.

Supported platforms

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Lumberyard to develop games for the following platforms:

- Microsoft Windows PC
- Android
- iOS
- macOS
- PlayStation
- Xbox
- Linux (dedicated server only)

Some platforms have additional requirements.

- For console support, see Developing Games for Xbox One and PlayStation Partners portals.
- For mobile devices, see Developing for Android and iOS with Lumberyard (p. 3166).
- For macOS, see Create macOS projects in Lumberyard (p. 3263).
- For Linux dedicated servers, see Creating Lumberyard Executables for Linux (p. 3269).

Setting up Amazon Lumberyard

Amazon Lumberyard has a two-part installation process:

1. The installer application downloads, extracts, and installs Lumberyard.
2. Setup Assistant configures and installs additional software based on your development needs. Jump right in to the Lumberyard Editor with an Express Install or customize Lumberyard's features for your development needs with a Custom Install.

   **Note**
   To create and build new projects, or add gems to existing projects, either select Custom Install or run Setup Assistant again later to enable this functionality.

After Lumberyard is installed, use Project Configurator to select a project or create a new project. The following sections detail the minimum requirements for Lumberyard and guide you through the installation process.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard has a minimum set of system requirements for development, as outlined in the following sections. Disk space and RAM requirements are dependent on the options that you choose during installation. If you have already installed Lumberyard, you can use Lumberyard Setup Assistant to see the requirements for the tools and features that you want for your Lumberyard development environment. Lumberyard Setup Assistant provides information about installing third-party software and SDKs on your computer. For more information, see Using Lumberyard Setup Assistant to Set Up Your Development Environment (p. 16).

If your system is capable of running a modern real-time 3D game with good performance you should be set, however, review these detailed requirements to be certain.

Lumberyard requires Windows 10.

**Lumberyard minimum hardware requirements:**

- 3 GHz quad-core processor
- 8 GB RAM
- 2 GB VRAM DirectX 11 or later compatible video card
  - NVIDIA GeForce GTX 660 Ti with driver version 368.81 or later
  - AMD Radeon HD 8730M with driver version 16.15.2211 or later
- 1366 x 768 px screen resolution
- 60 GB of free disk space

**Note**

If you select options to build the engine, editor, or tools in **Setup Assistant**, 14 GB RAM is required for compilation.

Some advanced graphics features require a DirectX 12 or later compatible video card. Required free disk space is dependent on the options that you select when installing Lumberyard.
Developer tools

You can use the Lumberyard Editor and tools without installing additional software. To create new projects or use advanced development features in Lumberyard, you need a developer environment that includes Microsoft Visual Studio. The following versions are supported:

- (Recommended) Microsoft Visual Studio 2019 version 16.2.4 thru version 16.9.x.
  
  **Important**
  There are some issues that arise when using the MSVC compiler shipped with Visual Studio 2019 versions 16.8, 16.8.1, and 16.8.2. For more details, check out the Lumberyard 1.28 Known Issues.
  

  **Note**
  The definitive source for the minimum version of Visual Studio for any particular release of Lumberyard is set in the file `dev\_WAF\settings\platforms\platform.win_x64_vs2019.json`. Look for the `vswhere-args` settings. The minimum version is the first value in the `default_value` range.

Microsoft offers Visual Studio Community edition free to individual developers. For more information and to download and install Visual Studio Community, visit the Visual Studio Community portal.

Visual Studio 2017 and 2019 required features

The default Visual Studio installation might not include all of the features that are required by Lumberyard. Ensure that the following Visual Studio features are enabled:

1. Launch the Visual Studio Installer from your download directory or the Start Menu if you've already installed Visual Studio.
2. If you've installed Visual Studio, choose More - Modify on the version of Visual Studio you'll use with Lumberyard.
3. On the Workloads tab:
   - Select Game development with C++.
   - In the Installation details panel on the right, select at least one Windows 10 SDK.
   - Select Desktop development with C++.
4. On the Individual components tab, under Compilers, build tools, and runtime, make sure that a VC++ toolset that corresponds to the installed version of Visual Studio has been selected:
   - If using Visual Studio 2017: Select at least one version of the VC++ 2017 toolset.
   - If using Visual Studio 2019: Select at least one version of the MSVC v142 - VS 2019 C++ x64/x86 build tool.
   - (Optional) To build with the Visual Studio 2017 toolset in Visual Studio 2019, select MSVC v141 - VS 2017 C++ x64/x86 build tools.

  **Note**
  Incredibuild users: Installing, reinstalling, or upgrading Visual Studio may cause the Incredibuild Agent to lose its settings or require an update. After modifying Visual Studio, be sure to confirm your Incredibuild Agent is active. For instructions on configuring the agent properly, see Compiling with IncrediBuild (p. 118).

  **Note**
  Beginning with Visual Studio 2017, Microsoft now releases updates on a more frequent cadence (in some cases weekly). Lumberyard is tested with the latest version of Visual Studio available during the release cycle.
Visual C++ redistributable packages

Lumberyard Installer will attempt to download and install the required Visual C++ Redistributable packages during the installation process. Under some circumstances, the installation of these redistributable packages may fail. If you receive a missing Visual C++ runtime .dll error while running Lumberyard Installer, Lumberyard Setup Assistant, or Project Configurator, do the following:

1. Check that the Visual C++ redistributable installers for Visual Studio 2012 and Visual Studio 2019 have been successfully downloaded. The installers are located in the corresponding Visual Studio directories in the \lumberyard_version\dev\Tools\Redistributables\ directory.
2. If the Visual C++ redistributable installers for Visual Studio 2012 and Visual Studio 2019 have not been downloaded by Lumberyard Installer, manually download the installers from Microsoft.
   • Visual C++ Redistributable for Visual Studio 2012
   • Visual C++ Redistributable for Visual Studio 2019
3. Run both Visual C++ redistributable installers.
4. Retry the Lumberyard installation after the Visual Studio redistributables have successfully installed.

Note

Installing Lumberyard

Download the latest version of Lumberyard Installer

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can also download earlier versions of Lumberyard and assets from the Amazon Lumberyard Downloads page.

Note
Verify that you have the required hardware and software for Lumberyard. For more information, see System requirements (p. 8).

For an in-depth tutorial about downloading and installing Lumberyard, watch the following video:

How to Install the Amazon Lumberyard Game Engine

Topics
• Using the Lumberyard Installer (p. 10)
• Using GitHub to Download Lumberyard (p. 11)
• Lumberyard Directories and Files (p. 12)

Using the Lumberyard Installer

The Lumberyard Installer extracts the Lumberyard files and adds shortcuts to your desktop and start menu for the following:
• Lumberyard Setup Assistant
• Project Configurator
• Lumberyard Editor

If you have an existing Lumberyard project and you want to upgrade, see Upgrading Lumberyard (p. 12).

Note
If you get file errors during extraction about AssetProcessor.exe, AssetProcessor_temp.exe, or CrySystem.dll, verify if your antivirus software is placing these files in quarantine. If needed, grant exceptions for the affected files.

To use the Lumberyard installer

1. Navigate to your download directory and run the Lumberyard Installer executable file: LumberyardInstaller.exe

2. (Optional) Change the location of the installation by selecting Options. The default installation location is C:\Amazon.

3. On the Welcome page of the installer, choose Install.

Note
In some situations, the Lumberyard Installer can stop responding and refuse to progress. The only known remedy at this time is to forcibly terminate the installer through the Windows Task Manager, or reboot, then retry the installation.

4. Follow the instructions to complete your installation.

5. On the Installation Successfully Completed page, click Launch Lumberyard Setup Assistant to install required third-party software and SDKs. For more information, see Running Lumberyard Setup Assistant (p. 16).

Using GitHub to Download Lumberyard

Each Lumberyard release exists as a separate branch in GitHub. To download Lumberyard, see the Amazon Lumberyard page on GitHub.
Lumberyard Directories and Files

After you install Lumberyard, the Lumberyard root directory includes the following folders and files:

- dev
  - _WAF_ – Waf build system files.
  - Bin64 – Binaries directory and configuration files for the Resource Compiler.
  - Bin64vc141 – Binaries directory and configuration files for Visual Studio 2017.
- Code – Source files directory and solution files.
- Editor – Editor assets.
- Engine – Engine assets.
- Gems – Optional systems and assets.
- MultiplayerSample – Multiplayer sample project that demonstrates how to build a multiplayer game with the new component entity system. For more information, see Multiplayer Sample (p. 155).
- ProjectTemplates – Configuration files, libraries, and scripts for the empty template.
- SamplesProject – Sample project. For more information, see Samples Project (p. 146).
- Tools – Third-party tools and plugins.
- engineroot.txt – System file required by Lumberyard Setup Assistant.
- 3rdParty
  - Third-party software required to use or compile Lumberyard
  - 3rdParty.txt – System file used by other third-party tools to verify the directory.

Upgrading Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you have an existing version of Lumberyard installed on your computer, you have several options for upgrading:

- Upgrade Lumberyard with an existing version in source control
- Upgrade Lumberyard without an existing version in source control
- Upgrade Lumberyard without source control

We recommend using source control, which allows relationships to be created between the installed versions of Lumberyard and the changes you make to your projects, among other benefits like revision history.

**Note**
When choosing a source control solution, keep in mind that Lumberyard provides plugins and tools for working with Perforce.

Topics

- Upgrading Lumberyard with an Existing Version in Source Control (p. 13)
- Upgrading Lumberyard without an Existing Version in Source Control (p. 13)
- Upgrading Lumberyard without Source Control (p. 13)
To set up Lumberyard in source control

1. Download and install Lumberyard. For information, see Installing Lumberyard (p. 10).
2. Check into source control a pristine, unmodified version of Lumberyard. For information about file types to ignore, see Files to Exclude When Upgrading Lumberyard (p. 15). For information about the Lumberyard directory structure, see Installing Lumberyard (p. 10).
3. In source control, create a new branch off the pristine Lumberyard branch to use for development.
4. Make changes to the new development branch only.

Upgrading Lumberyard with an Existing Version in Source Control

Before you begin upgrading, check into source control the previous pristine version of Lumberyard.

To upgrade Lumberyard with an existing version in source control

1. In Windows Explorer, locate the directory where you installed the previous pristine version of Lumberyard. Delete the contents of this directory to remove the files from source control.
2. Download and install the new version of Lumberyard to the empty directory. Ensure the directory structure is identical to the previous version.
3. Using source control, reconcile the files in the directory with the files in the pristine Lumberyard branch. For example, if you use Perforce, click Actions, Reconcile Offline Work.
4. Build and test the reconciled version locally to ensure it works.
5. Submit the reconciled version to the pristine Lumberyard branch as the new version of Lumberyard.
6. Integrate the updated, pristine Lumberyard branch into your development branch.

Upgrading Lumberyard without an Existing Version in Source Control

Follow these steps to prepare your source control to upgrade Lumberyard.

To upgrade Lumberyard without an existing version in source control

1. Check into source control the pristine version of Lumberyard that you used to create your game project.
2. Create a new branch off the pristine Lumberyard branch to use for development.
3. In Windows Explorer, locate the directory for the new development branch and delete the contents.
4. Copy the files from your existing game project to the empty directory.
5. Using source control, reconcile the files in the development branch directory with the files in source control. Accept your changes.
6. Follow the steps in Upgrading Lumberyard with an Existing Version in Source Control (p. 13).

Upgrading Lumberyard without Source Control

You can upgrade Lumberyard without using source control; however, we do not recommend this method.
To upgrade Lumberyard without source control

1. Download and install the latest version of Lumberyard to a location that will not overwrite any previous versions. For information, see Installing Lumberyard (p. 10).
2. Use Lumberyard Setup Assistant to install the third-party software and SDKs required to run Lumberyard. For information, see Running Lumberyard Setup Assistant (p. 16).
3. Configure and compile the Samples Project to test your build environment.

Upgrading Your Game Projects

Once you have upgraded Lumberyard, you can upgrade each of your game projects.

To upgrade your game project

1. Copy your project's code (located in the \lumberyard_version\dev\Code\project_name directory) and game folder (located in the \lumberyard_version\dev\project_name directory) to the new Lumberyard directory.
2. Create a project.json file for your project with the following:

   ```json
   {
     "project_name": "project_name",
     "product_name": "project_name",
     "executable_name": "project_nameLauncher",
     "code_folder": "Code/project_name",
     "modules": ["project_name"]
   }
   ``

   Replace all instances of project_name with your project's name.

   For example, if your project was called MyProject, the project.json file would include the following:

   ```json
   {
     "project_name": "MyProject",
     "product_name": "MyProject",
     "executable_name": "MyProjectLauncher",
     "code_folder": "Code/MyProject",
     "modules": ["MyProject"]
   }
   ``

3. Save the project.json file in the \lumberyard_version\dev\project_name directory.
4. Run the Project Configurator (located in the \lumberyard_version\dev\Bin64 directory) and set your game project as the default project. When finished, close the Project Configurator.
5. Edit the wscript file (located in the \lumberyard_version\dev\Code\project_name\Game directory) to ensure the includes under #Common appear as follows:

   ```
   #===============================================================================
   # Common
   #===============================================================================
   includes = [ '.', 
                bld.Path('Code/CryEngine/CryCommon'),
                bld.Path('Code/CryEngine/CryAction')],
   ```

6. In a command line window, locate the new dev directory and enter the following build command for your version of Visual Studio:
Files to Exclude When Upgrading Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When adding Lumberyard to source control, there are various files that you should exclude because they are generated, temporary, or programming-specific.

File types and folders in the entire repository to exclude

- *.ilk
- *.suo
- *.user
- *.o
- *.temp
- *.bootstrap.digests
- *.log
- *.exp
- *.vssettings
- *.exportlog
- *.mayaSwatches
- *.ma.swatches
- *.dds
- *.bak
- *.bak2
- *.options
- *.pyc
- *.db
- Solutions
- BinTemp
- Cache

File types and folders in the \dev\Code directory to exclude.

- SDKs

File types and folders in each game folder (SamplesProject, MultiplayerProject, and so on) to exclude.

- Compiled assets
  - *.dds
  - *.caf
  - *.animsettings
- Editor backup files – *.bak*
Set Up a Development Environment

- Pak files that are exported from level files in the editor – *.pak

Using Lumberyard Setup Assistant to Set Up Your Development Environment

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Lumberyard Setup Assistant application to validate that you have installed the third-party software required to run Lumberyard.

Lumberyard Setup Assistant offers the following benefits:

- Ensures you have the required runtime software installed
- Ensures you have the required SDKs located in the source tree
- Provides plugins for certain programs detected
- Validates registry settings, paths, and libraries

You should run this application periodically and after you make any changes to your environment, to validate and repair settings and paths. You can also customize the application with a configuration file to easily integrate your specific directory structure.

**Note**
Certain third-party software may require a license. Please consult the terms of service before installing the software.

Prerequisites

- Lumberyard Setup Assistant is supported on the Windows operating system.
- Certain options require an installation of Microsoft Visual Studio. For more information, see System requirements (p. 8).

Topics

- Running Lumberyard Setup Assistant (p. 16)
- Using Lumberyard Setup Assistant Batch (p. 18)
- Customizing Lumberyard Setup Assistant (p. 20)

Running Lumberyard Setup Assistant

To learn about configuring Lumberyard Setup Assistant, watch the following video tutorial.

Note
Before you run Lumberyard Setup Assistant, verify the following:
• 3rdParty.txt file appears in the `lumberyard_version\3rdParty` directory
• engineroot.txt appears in the `lumberyard_version\dev` directory

Lumberyard Setup Assistant requires these files to properly detect third-party software and SDKs.

**To use Lumberyard Setup Assistant**

1. Open Lumberyard Setup Assistant with your preferred method:
   - From the desktop, double-click the **Setup Assistant** icon
   - Navigate to the `lumberyard_version\dev\Tools\LmbrSetup\Win` directory and double-click `SetupAssistant.exe`

2. In the **Full Install** box, choose **Start a Full Installation**.

3. Verify that the engine root path is correct.

4. On the **Get started** page, select what you want to do:
   - **Create, Modify and Build projects** – Select this option if you intend to create and build new projects with Lumberyard on the **Windows platform**, or make changes to the engine, asset pipeline, or other tools.
   - **Compile for Android devices**
   - **Setup for Linux Dedicated Server**

You can also create, enable, and disable these capabilities from the command line. For more information, see Using Lumberyard Setup Assistant Batch (p. 18) and Lmbr.exe (p. 56).

5. Select **Visual Studio 2017** or **Visual Studio 2019**.

**Note**

We recommend that you select only **one** version of Visual Studio - the one that you will use to compile your projects. If both are enabled, you will be asked which compiler to use when you rebuild your project in Project Configurator.
**Set Up a Development Environment**

**Note**
Selecting a version of Visual Studio here will enable it as a build platform and enable a Visual Studio solution to be generated for that version. For more information about build settings specific to Visual Studio, see Waf User Options and Settings (p. 78).

**Note**
For more information about installing and configuring Visual Studio for Lumberyard, see Developer tools (p. 9).

6. Click **Next**.
7. Follow the instructions on each page.
8. When you have completed installing software and SDKs, the **Summary** page displays information about your Lumberyard environment. From the **Summary** page, you can launch the Lumberyard Editor by choosing **Launch Editor**. If you’d like to choose an existing project or create a new project, choose **Configure project** to launch Project Configurator. For more information about configuring your project, see Creating Lumberyard projects (p. 43).

### Using Lumberyard Setup Assistant Batch

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The command line version of Lumberyard Setup Assistant is useful for server and build administrators who want to create a batch file to run the same configuration on multiple machines.

You can find the executable file `SetupAssistantBatch.exe` in the `lumberyard_version\dev\Tools\LmbrSetup\Win` directory. This executable uses the `SetupAssistantConfig.json` file to complete its tasks.

You can also use the batch file to create, enable, and disable capabilities, and to modify your projects and gems. For more information, see `Lmbr.exe` (p. 56).

**To use Lumberyard Setup Assistant Batch**

1. Open a command line window.
2. Navigate to the `Win` directory for your Lumberyard installation.
   ```bash
cd lumberyard_version\dev\Tools\LmbrSetup\Win
   ```
3. Run the `SetupAssistantBatch.exe` and modify the commands as needed. See the following list of commands.

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--help</code></td>
<td>Lists all commands and descriptions.</td>
</tr>
<tr>
<td><code>--3rdpartypath</code></td>
<td>Sets the third-party directory to the specified parameter. The default is the <code>lumberyard_version\3rdParty</code> directory. If you change the third-party directory, ensure that you also move the <code>3rdParty.txt</code> file.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--sdkpath</td>
<td>Sets the location of the Lumberyard SDK to the specified parameter.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This command expects a root where Lumberyardroot.txt is located.</td>
</tr>
<tr>
<td>--disablecapability</td>
<td>Disables the specified capabilities.</td>
</tr>
<tr>
<td></td>
<td>• Run your game project.</td>
</tr>
<tr>
<td></td>
<td>• Run Lumberyard Editor and tools.</td>
</tr>
<tr>
<td></td>
<td>• Compile the game code.</td>
</tr>
<tr>
<td></td>
<td>• Compile the engine and asset pipeline.</td>
</tr>
<tr>
<td></td>
<td>• Compile Lumberyard Editor and tools.</td>
</tr>
<tr>
<td></td>
<td>• Compile for Android devices.</td>
</tr>
<tr>
<td></td>
<td>• Compile for iOS devices.</td>
</tr>
<tr>
<td>--enablecapability</td>
<td>Enables the specified capabilities:</td>
</tr>
<tr>
<td></td>
<td>• Run your game project.</td>
</tr>
<tr>
<td></td>
<td>• Run Lumberyard Editor and tools.</td>
</tr>
<tr>
<td></td>
<td>• Compile the game code.</td>
</tr>
<tr>
<td></td>
<td>• Compile the engine and asset pipeline.</td>
</tr>
<tr>
<td></td>
<td>• Compile Lumberyard Editor and tools.</td>
</tr>
<tr>
<td></td>
<td>• Compile for Android devices.</td>
</tr>
<tr>
<td></td>
<td>• Compile for iOS devices.</td>
</tr>
<tr>
<td>--all</td>
<td>Enables all capabilities.</td>
</tr>
<tr>
<td>--none</td>
<td>Disables all capabilities.</td>
</tr>
<tr>
<td>--no-modify-environment</td>
<td>Prevents Lumberyard Setup Assistant from changing your environment variables.</td>
</tr>
</tbody>
</table>

**Example Commands**

The following example sets the paths, clears all selected capabilities, and then enables the capability rungame.

```bash
setupassistantbatch.exe --3rdpartypath "d:\lumberyard_version\3rdParty" --sdkpath "d:\lumberyard_version\dev" --none --enablecapability rungame
```

**Note**

To examine the values for each capability, see the SetupAssistantConfig.json file in the lumberyard_version\dev directory.

If the command runs correctly, this program returns the exit code 0.

The following example disables all capabilities and enables only the compilation tasks. This command is commonly used for hosting a build server:

```bash
setupassistantbatch.exe --none --enablecapability compilegame --enablecapability compileLumberyard --enablecapability compilesandbox
```
Customizing Lumberyard Setup Assistant

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the SetupAssistantConfig.json configuration file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory) to customize Lumberyard Setup Assistant for your project. The settings in this file are prioritized above internal default settings.

For example configuration data that you can copy and paste into your own .json file, refer to the SetupAssistantConfig.json.example file (located in the \lumberyard_version\dev\Tools \LmbrSetup directory).

Save your changes to the SetupAssistantConfig.json configuration file in the \lumberyard_version\dev\Tools\LmbrSetup directory. To validate your changes and ensure there are no syntax errors (such as a missing comma), run the SetupAssistantBatch.exe file (located in the \lumberyard_version\dev\Tools\LmbrSetup\Win directory) from a command line window.

Enabling and Disabling Features

You can enable or disable software and third-party SDKs based on your project requirements. In the SetupAssistantConfig.ini file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory), uncomment the lines (delete the semicolon) to disable specific features.

Adding New Third-Party SDKs

In addition to enabling or disabling software and third-party SDKs, you can edit the SetupAssistantConfig.json file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory) to add new, third-party SDKs to your project configuration.

When you add third-party SDKs to the SetupAssistantConfig.json file, which is loaded after the internal configuration file, the .json file removes and replaces entries in the internal configuration. This allows you to customize your project configuration without having to recompile.

To add new tasks (capabilities)

1. If currently running, close the Lumberyard Setup Assistant.
2. Navigate to the \lumberyard_version\dev\Tools\LmbrSetup directory.
3. Edit the SetupAssistantConfig.json file to do the following:
   a. Add the task(s) to the Capabilities section.
   b. Update the third-party SDKs to include the appropriate tags.
Set Up a Development Environment

4. Open the Lumberyard Setup Assistant to see your changes.

To add third-party SDKs

1. If currently running, close the Lumberyard Setup Assistant.
2. Navigate to the \lumberyard_version\dev\Tools\LmbrSetup directory.
3. Edit the SetupAssistantConfig.json file to add information for your third-party SDKs. See SDK Fields (p. 21).
   
   Note
   For an example of how to add configuration data, refer to the SetupAssistantConfig.json.example file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory).

4. Open the Lumberyard Setup Assistant to see your changes.

To remove existing third-party SDKs

1. If currently running, close the Lumberyard Setup Assistant.
2. Navigate to the \lumberyard_version\dev\Tools\LmbrSetup directory.
3. Edit the SetupAssistantConfig.json file to create a remove entry with the same identifier.
   
   Note
   For an example of how to add configuration data, refer to the SetupAssistantConfig.json.example file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory).

4. Open the Lumberyard Setup Assistant to see your changes.

Note
When you specify the destination of your code directory, you can use $CODEFOLDERNAME$ or specify the actual name. The code directory is the location where third-party SDKs are expected and is relative to the SDK root. For example, you can change CodeFolderName to myGame/A/b/c.

SDK Fields

You may need to provide information for the following SDK fields.

identifier

Unique string identifier that refers to the SDK. The identifier must be one word and use only lowercase letters.

remove

If true, eliminates an existing entry. The remove and identifier fields are required to remove an entry.

name

Display name of the SDK. Custom SDKs can use any name without language restrictions.
Lumberyard User Guide Lumberyard User Guide
Set Up a Development Environment

version
Version of the SDK.
description
Brief description of the SDK. Custom SDKs can use any description. UTF-8 is supported.
detailedInstructions
(Optional) Detailed instructions to obtain and install the SDK.
tags
Tags to which the SDK applies. For example, to have the SDK run the game, add the rungame tag.
symlinks
List of symlink dictionaries for all junctions (symbolic links) to establish between the 3rdParty directory and the code base. Each symlink uses the following form:
• source – Source directory, relative to the 3rdParty directory.
• destination – Destination directory, relative to the SDK root.
• exampleFile – File that is located in the source and destination folders and that validates the established link.

Configuring Advanced Settings
The SetupAssistantConfig.json file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory) has the following configuration settings in the root element (dictionary):

CodeFolderName
Location of the code directory, relative to the Lumberyardroot.txt file. You can specify relative paths such as .. and ../.. (use forward slash marks) or relative paths with multiple components such as code/mycode/stuff.

ToolsFolderName
Location of the tools directory, relative to the Lumberyardroot.txt file. The default directory is Tools, but you can specify relative folders such as ../tools.

RememberLumberyardRootFolder
If true, saves the Lumberyard root that the user browsed between sessions. If false, automatically detects the Lumberyard root based on the executable location. The default value is false.

Remember3rdPartyFolder
If true, saves the third-party directory that the user browsed between sessions. If false, automatically detects the third-party directory based on the executable location. The default value is false.

Customizing the Maya Environment
The \Tools\Maya\Plugins directory includes the Lumberyard Maya plugin, and the \Tools\Maya\script directory includes the MEL and Python scripts. To enable the Maya plugin functionality, Lumberyard Setup Assistant modifies Maya.ENV to add the required variables to your Maya configuration.

If you use your own Maya tools in addition to the exporter and pipeline tools that Lumberyard provides, you can use the SetupAssistantConfig.json file to add your project-specific paths to the MayaEnvironments tag.

For example configuration data, refer to the SetupAssistantConfig.json.example file (located in the \lumberyard_version\dev\Tools\LmbrSetup directory).
In the following example, #$TOOLSFOLDER#$ is a macro that is substituted with the appropriate tools directory. However, you can also use relative paths, relative to the game project’s root directory that includes Lumberyardroot.txt:

```
"MayaEnvironments" : 
  [ 
    "comment" : "an example entry showing how you can add a path to MAYA_PLUG_IN_PATH in maya.env",
    "identifier" : "MAYA_PLUG_IN_PATH",
    "paths" : ['$TOOLSFOLDER$/maya/plugins']
  ],
  [ 
    "comment" : "an example entry showing how you can add paths to MAYA_SCRIPT_PATH in maya.env",
    "identifier" : "MAYA_SCRIPT_PATH",
    "paths" : [ "%DHTECH_SCRIPT_PATH%\%DHTECH_GAME_PATH%", "%DHTECH_SCRIPT_PATH%\animation" ]
  ],
],
```

### Updating the Code or Tools Location

If your project requires moving the Lumberyard code or tools directory so that it's no longer located in a subfolder called Code or Tools relative to the Lumberyard root, you can edit the SetupAssistantConfig.json file (located in the \lumberyard_version\dev\Tools \LmbrSetup directory) to update the directory location. Ensure the updated directory includes the Lumberyardroot.txt file.

### Enabling a Firewall

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

You can help protect your environment by enabling the firewall settings on all computers running the Asset Processor or Lumberyard Editor to do the following:

- Exclude external connections to ports 4600, 9432, 9433, and 45643 from untrusted IP addresses.
- Exclude connections from every address except 127.0.0.1.
- If you have multiple computers that work together (e.g. a PC and a Mac), you must allow connections to ports 4600, 9432, 9433, and 45643 from the IP addresses for these computers, but exclude all other connections.

Refer to the documentation for your operating system for how to manage your firewall settings.

### Using the Perforce Plugin with Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard integrates with Perforce as a source control solution. The engine uses the `p4 set` command to configure settings, and the Perforce visual client (P4V) to selectively sync and submit changed assets.
You can use the **Perforce Settings** dialog box to configure how Lumberyard connects to Perforce. The following settings are cached and populated when the dialog box opens:

- **Server** (P4PORT)
- **User** (P4USER)
- **Workspace** (P4CLIENT)
- **Charset** (P4CHARSET)

  **Note**
  
P4_<P4PORT>_CHARSET is also cached. This value is used if it matches your current P4PORT value; otherwise, the value for P4CHARSET is used. For example, if P4PORT is set to my.perforce.server.com:1666, the value P4_my.perforce.server.com:1666 would be used.

Certain values may not be modifiable if your Perforce connection settings are configured using a method that overrides the p4 set command. The following connection methods may override the ability to modify a setting:

- **Config** – A configuration file overrides this connection setting. If detected, the path to the configuration file is displayed. If undetected, you can check the setting for P4CONFIG.
- **Environment** – Your system environment overrides this connection setting. You can check your system's control panel to remove these overrides.

**To use the Perforce plugin menu**

1. In Lumberyard Editor, click the P4 icon in the bottom toolbar.

   **Note**
   
   Hover over the icon to display the connection status.

2. In the drop-down menu, you can do the following:

   - Click **Enable** or **Disable** to toggle the plugin. The **Enable** setting allows you to work online. The **Disable** setting forces you to work offline.

     **Note**
     
     Changes are not tracked in offline mode. If you work offline, you must manually reconcile your work when you reconnect to Perforce.

   - Click **Settings** to view or modify your Perforce settings.

   To restore default settings, click **Reset** for each value. When finished, click **OK** to apply your changes.

---

**Uninstalling Amazon Lumberyard**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you want to uninstall Lumberyard, you can remove it with Windows Control Panel. The uninstall removes the Lumberyard Setup Assistant, Project Configurator, and Lumberyard Editor.

**To uninstall Lumberyard**

1. Go to the **Control Panel** and navigate to the list of programs on your computer.
2. Click Lumberyard_**your_version** and then choose Uninstall.
3. In the Lumberyard Setup prompt, choose Uninstall.
   
   After the process is done, the **Uninstall Successfully Completed** message appears.

---

## Lumberyard Programming Concepts and Resources

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This topic provides brief explanations of some important Lumberyard programming concepts and resources. More information links are provided.

**Topics**
- Asset System (p. 25)
- AZ Code Generator (p. 25)
- Building (p. 26)
- Cloud Canvas (p. 26)
- Debugging (p. 26)
- EBuses (p. 26)
- Entities and Components (p. 26)
- Gems and AZ Modules (p. 27)
- Input (p. 27)
- Lua (p. 28)
- Networking (p. 28)
- Reflection System (p. 28)
- Script Canvas (p. 28)
- Slices (p. 28)
- C++ Best Practices for Amazon Lumberyard (p. 28)

### Asset System

The Lumberyard Editor and Lumberyard runtime code use the Lumberyard asset system (p. 247) to asynchronously stream and activate assets.

For information on using the runtime asset system to load already-built assets into a running instance of the engine, see Programming the Lumberyard AZCore Runtime Asset System (p. 304).

For information on adding a custom asset type to Lumberyard, see Adding an Asset Type to Lumberyard (p. 308).

For information on creating a builder for custom assets, see Creating a Custom Asset Builder (p. 318).

### AZ Code Generator

You can use the AZ Code Generator command line utility to generate boilerplate code when the structure of the intended code is known in advance. AZ Code Generator generates source code (or any data or text) from source code that is specially tagged.
Building

You can build your Lumberyard projects in profile mode, debug mode, or release mode.

For more information, see Building Lumberyard projects (p. 61).

Cloud Canvas

Lumberyard's Cloud Canvas (p. 2106) enables connected game features that use AWS cloud computing. You can use the cloud gems (p. 2127) that are included with Lumberyard to provide prepackaged cloud-connected game features. You can use the Cloud Gem Framework (p. 2257) to create your own cloud gems (p. 2259).

For more information about programming with Cloud Canvas, see Getting Started with the Cloud Gem Framework (p. 2258).

Debugging

Lumberyard includes a number of programming tools for testing, profiling, and debugging.

For more information, see Profiling, Testing, and Debugging Game Projects (p. 1920).

EBuses

EBuses (event buses) are Lumberyard's general-purpose messaging system. They dispatch notifications and receive requests.

Components commonly use EBuses in two ways: to dispatch events from a notification bus or to handle requests using a request bus. Some components provide one type of bus, and some components provide both types. You use the EBus class for both EBus types. Some components do not provide an EBus at all.

To interact with the engine or other components in Lumberyard, include the component or system's EBus or API header in your code. Then make calls to the exposed EBuses. With this approach you can replace engine level system APIs with implementations that you define in a gem. For example, you could replace Lumberyard’s audio system with your own EBus handler. This would give you complete control over audio without having to recompile the engine.

For information about EBuses, see Working with the Event Bus (EBus) system (p. 1851).

For information about using EBuses in Lua, see Using EBuses in Lua (p. 2693).

Entities and Components

Lumberyard uses a lightweight entity/component model called the component entity system for both game objects and systems. Lumberyard entities are simply an ID and a container of components. They have no functionality associated with them. Lumberyard's component model is granular: It expects each component to provide independent functionality. It expects game objects to be made up of one or more entities with many components attached to each entity. Lumberyard components communicate with each other using a messaging system called EBuses (explained later). Lumberyard expects you to use EBuses instead of holding references to other entities or their components.

Lumberyard's components have a simple lifecycle. When an entity is activated, it calls Activate() on all of its components. When the entity is deactivated, it calls Deactivate() on all of its components. In the Activate() function a component sets itself up, connects to EBuses, and allocates resources.
or requests assets. In the `Deactivate()` function, a component should release all resources and disconnect from all EBuses. Components should be completely dormant after deactivation, and they should be in more or less the same state that they are in after `Init()` is called. The `Init()` function is only called once and allows a component to initialize its internal state. Note that a component can be activated and deactivated many times before it is deleted. For example, it might be deactivated temporarily while it is being streamed out.

The remaining API operations of a component should be established by the EBus that it implements.

Components can depend on services. Such services usually have a one-to-one relationship with EBuses. If a component declares that it depends on (requires) a service, any entity that uses the component must also contain a component that provides the required services.

Components are always activated in order of their dependency. For this reason, a component can always assume that the services that it requires are available when the component is activated.

Entities are never allowed to be in a state in which one of their component dependencies is missing, even during authoring. In Lumberyard's editing tools, this means that components whose dependencies are missing are actually removed from the entity and stored in a temporary list. When all of the component's dependencies become available, the component is restored to the entity.

For more information about entities and components, see Programmer's Guide to Entities and Components (p. 973).

## Gems and AZ Modules

Lumberyard is designed to create applications that are small executables. A Lumberyard application has a simple application class (`AZ::ComponentApplication`). The application class reads a manifest (a `gems.json` file) and loads modules that provide functionality for your game. Lumberyard calls these modules gems. Lumberyard's goal is to have you choose a set of gems that match the requirements of your game and only compile what you need. Lumberyard includes a variety of prebuilt gems that add functionality to the game engine, such as VR, cloud connectivity, and the EMotion FX animation system.

Cloud gems (p. 2127) are gems that add AWS cloud-connected functionality to your game.

When you create a game, your game code and assets go into one or more gems. Each code gem contains an `AZ::Module` (p. 1090). An AZ module is a collection of C++ code built as a static or dynamic library (.lib or .dll file) that implements specific initialization functions. The AZ module is the interface that Lumberyard uses to extract the contents of your gem into the global environment.

Each application has a single entity associated with it that is referred to as the **system entity** (p. 1104). AZ modules can add components to this entity before it is activated.

Components that are added to the system entity are called **system components** (p. 1098). System components are often singleton/manager-type objects that aggregate or provide resources to game components or other systems. Like other entities, a system entity must have its dependencies present. You can assume that any systems that you depend on are booted and available when your system component is activated.

For information on programming with gems, see Programming with Gems (p. 1073).

For information on creating system components, see Creating System Components (p. 987).

## Input

Lumberyard provides independent, extensible, and efficient access to input data from supported operating systems and devices. Lumberyard's AZ framework input interface provides abstracted interfaces for OS-specific features like file I/O and application lifecycle management.

For more information, see Input in Amazon Lumberyard (p. 1902).
Lua

Lua is a powerful, fast, lightweight, scripting language. You can use Lua in Lumberyard to facilitate quick iteration of your game project. When you construct new gameplay and game systems, you can run your changes immediately, without compiling your source code.

For more information on using Lua in Lumberyard, see Writing Lua Scripts (p. 2683).

Networking

Lumberyard's networking system is designed for efficient bandwidth usage and low-latency communications. It uses a replica framework to synchronize objects over the network, and its session management integrates with major online console services. Lumberyard networking lets you handle peer-to-peer and client-server topologies with host migration and supports in-game achievements, leaderboards, and cloud-based saved games.

For more information, see Using Lumberyard Networking (p. 1990).

Reflection System

You can use Lumberyard's reflection system to expose runtime code for C++-based objects, for Lumberyard Editor, and for scripting (Lua and Script Canvas). Lumberyard provides three reflection contexts for this purpose: a serialization context (p. 1033), a behavior context (p. 1053), and an edit context (p. 1052).

For more information, see Reflecting Lumberyard Classes, Methods, and EBus Interfaces (p. 1032).

Script Canvas

Script Canvas (p. 2420) is Lumberyard's visual scripting environment.

For information on creating your own nodes in Script Canvas, see Creating Custom Nodes in Script Canvas (p. 2553).

For information on using the behavior context to expose runtime code to Script Canvas, see Script Canvas and the Behavior Context (p. 2543).

Slices

Slices are practical and powerful way to create units of content in Lumberyard and manage them. A slice is a collection of one or more entities. You can instantiate a slice as many times as you require. Many game engines use levels and/or sublevels for their content, but Lumberyard loads content into its engine in the form of slices. Because slices can inherit from each other and be nested, they are powerful tools for managing content.

Like most systems that use prefabs or archetypes, you can use inheritance to override properties in Lumberyard. However, you can use Lumberyard's slices to add or remove components or even entire entity hierarchies.

For more information, see Working with Slices (p. 510).

C++ Best Practices for Amazon Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The following topics present some best practice coding patterns that Lumberyard engineers have found helpful for writing high quality code. Because of the complexity of C++ and the variety of ways it is used in Lumberyard, consider these guidelines as suggestions rather than a formal standard.

**Topics**

- Memory Management (p. 29)
- Use AzCore Standard Containers (p. 31)
- Tracing (p. 31)

## Memory Management

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When managing memory in Lumberyard, use AZ memory management calls and avoid static variables whose constructors allocate memory or connect to EBuses.

### Memory Allocations

When allocating memory, use the following recommended practices:

- **Do not use** `new`, `malloc`, and similar allocators directly. Instead, use AZ memory manager calls to `aznew`, `azmalloc`, `azfree`, `azcreate`, and `azdestroy`.
- **Specify the allocator in each class.**
- **Use child allocators.** To tag and track resource usage, new gems and subsystems should create their own allocator or create a `ChildAllocator` that references an existing allocator. For an example of creating a child allocator, see Creating an Allocator (p. 1911).

**Reason:** Lumberyard's core AZ systems provide a memory managed environment, not a raw system allocator like the managed memory in C# or Java. In Lumberyard, the core AZ systems provide speed, safety, and facilities for tracking memory usage.

For information about Lumberyard's scheme for allocators and new allocators, see Manually Allocating Memory (p. 1907).

### Memory Issues Caused by Static Variables

In constructors and destructors for static variables, avoid the following:

- Allocating memory with the AZ memory allocator system.
- Deallocating memory with the AZ memory allocator system.
- Connecting to EBuses
- Disconnecting from EBuses.

These rules apply to global static variables, function local static variables, and containers.

**Reason:** Lumberyard manages the memory of its classes and container types using the AZ memory allocator system. However, class static variables and containers can attempt to use the AZ memory allocator system before it is created or after it no longer exists. Keep in mind the following:

- On application startup or when a gem is loaded, if a static variable attempts to use the AZ system allocator before the AZ system allocator has been created, a crash can occur.
• On shutdown, if a static variable attempts to deallocate from the AZ system heap when the AZ system allocator no longer exists, a crash can occur.

Remember that the lifetime of a static variable lasts until the module in which the variable is declared is unloaded. In static linked libraries, the duration of static variables is the entire duration of the application.

Static Local Variable Example

The following example code from the Lumberyard version 1.17 AzFramework (\dev\Code\Framework\AzFramework\AzFramework\Physics\DefaultDebugDrawSettings.h) uses function static container variables whose destructors rely on AZ::SystemAllocator.

```
namespace Physics
{
    AZ_INLINE void GetDefaultDebugDrawSettings(DebugDrawSettings& settings)
    {
        static AZStd::vector<Vec3> cryVerts;      // These function static container variables use AZ::SystemAllocator.
        static AZStd::vector<ColorB> cryColors;
        // ...
    }
}
```

By default, the AZStd::vector class uses the SystemAllocator. Invoking this function on application shutdown, when SystemAllocator no longer exists, causes a deletion from a destroyed heap.

Global Static Variable Example

On startup, the following code attempts to create a global static variable whose constructor connects to an EBus:

```
// Example of a global static variable that connects to an EBus.
namespace LmbrCentral
{
    // Wraps an IMaterial pointer so that BehaviorContext can use it.
    class MaterialHandle : public AZ::RenderNotificationsBus::Handler
    {
        public:
            AZ_CLASS_ALLOCATOR(MaterialHandle, AZ::SystemAllocator, 0);
            AZ_TYPE_INFO(MaterialHandle, "{BF659DC6-ACDD-4062-A52E-4EC053286F4F}");
            MaterialHandle()
            {
                AZ::RenderNotificationsBus::Handler::BusConnect();
            }
            // ...

    // Later in the code, a MaterialHandle in global space is declared:
    MaterialHandle g_defaultMaterialHandle;
```

The code crashes the engine as soon as the module that contains the global variable definition is loaded. The constructor of the global variable attempts to connect to the EBus, but the EBus system is not ready because it uses an environment variable from a module that has not yet been initialized.
Use AzCore Standard Containers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Recommended:** Instead of writing your own containers, use the full range of AzCore standard containers in Lumberyard. Because these containers provide AZ memory allocators by default, they work with AZ memory manager out of the box. These containers include the following:

- array
- bitset
- deque
- fix_unordered_set
- fixed_forward_list
- fixed_unordered_map
- fixed_vector
- fixed_list
- forward_list
- list
- map
- multimap
- multiset
- queue
- ring_buffer
- set
- stack
- unordered_map
- unordered_multimap
- unordered_multiset
- vector

For multithreaded environments, Lumberyard has concurrent versions of `vector`, `map`, `set`, and `fixed`. For the source code, see `lumberyard_version\dev\Code\Framework\AzCore\AzCore\std\containers`.

**Recommended:** Create containers by value. Because none of the AZStd containers allocate memory when they are empty, creating containers with `new` on the heap is usually not necessary.

**Recommended:** Store container contents by value. If the container is the owner of dynamically allocated contents, store the contents in AZStd smart pointers.

### Tracing

**Recommended:** For code tracing needs, use `AZ_*` tracing macros.

**Reason:** Error handling and tracing functions provide useful messages formatted for readability about errors that occur. To facilitate debugging, `AZ_*` tracing functions indicate where in code the errors occurred.

The following table describes the `AZ_Tracing` macros and their uses.
<table>
<thead>
<tr>
<th>AZ Tracing Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ_Assert</td>
<td>Use for critical errors when the program cannot continue. AZ_Assert macros print an error message, the file name and line number where the error occurred, and then break program execution.</td>
</tr>
<tr>
<td>AZ_Error</td>
<td>Use when an obvious error has occurred but the program can continue safely. AZ_Error macros print an error message and the file name and line number where the error occurred. In some environments, AZ_Error notifies the user that an error has occurred (for example, in a message box or onscreen message).</td>
</tr>
<tr>
<td>AZ_Warning</td>
<td>Use when an error might have occurred. AZ_Warning macros print an error message and the file name and line number of the possible error, but take no other action.</td>
</tr>
<tr>
<td>AZ_TracePrintf</td>
<td>Use for informational purposes only. AZ_TracePrintf macros print a message but take no other action.</td>
</tr>
</tbody>
</table>

For source code, see `lumberyard_version\dev\Code\Framework\AzCore\AzCore\Debug\Trace.*`.

**Suppressing AZ Trace Messages in Unit Tests**

You can use macros to suppress AZ trace messages. Starting in Lumberyard version 1.21, the following macros for suppressing AZ Trace messages in the AZ Unit Test framework have equivalent macros with new names. The old macros will be removed in a future release.

<table>
<thead>
<tr>
<th>Old Macro Name</th>
<th>New Macro Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ_TEST_START_ASSERTTEST</td>
<td>AZ_TEST_START_TRACE_SUPPRESSION</td>
</tr>
<tr>
<td>AZ_TEST_STOP_ASSERTTEST</td>
<td>AZ_TEST_STOP_TRACE_SUPPRESSION</td>
</tr>
</tbody>
</table>

The old macros only suppressed the AZ_Error and AZ_Assert trace messages, but their names implied application termination. The new macros are equivalent in functionality to the old macros but have names that more accurately reflect their functionality.

For source code, see `lumberyard_version\dev\Code\Framework\AzCore\AzCore\UnitTest\UnitTest.h`.

### Lumberyard Editors and Tools

This topic references tools and features that are *legacy*. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the [AWS Game Tech blog](https://aws.amazon.com/pt/blogs/gaming/) to learn more.
Lumberyard provides the following suite of applications, editors, and tools for game development.

Animation Editor (p. 1340)

Manages character animations, attachments, and physics simulations along with blendspace and animation layering.

Asset Browser (p. 296)

Displays all game assets available for use.

Asset Processor (p. 248)

Runs in the background when you launch Lumberyard Editor, monitoring input directories for changes in source files and automatically generating platform-specific game assets as they change.

Audio Controls Editor (p. 3142)

Manages audio translation layer (ATL) controls and events for the audio system.

Animation Editor (Legacy)

Manages character animations, attachments, and physics simulations along with blendspace and animation layering.

Console (p. 210)

Runs editor commands and lists available console variables.

Database View (p. 1264)

Displays various object libraries such as entities, particles, and prefabs.

Entity Inspector (p. 475)

Displays the ID and name for component entity system objects.

Entity Outliner (p. 462)

Displays all component entities used for a level.

FBX Settings (p. 409)

Imports single static meshes and materials from FBX.

Layers (p. 469)

Organizes level data into discrete files.

Lens Flare Editor

Creates and manages camera lens flare effects.

Sun Trajectory Tool (p. 1265)

Creates and manages dynamic sky lighting effects.

LOD Generator

Generates geometry and material level of detail (LOD).

Lumberyard Editor (p. 182)

Acts as the main workspace editor and game viewport; loads the Rollup Bar and console by default.

Lumberyard Setup Assistant (p. 16)

Ensures that you have the necessary runtime software and SDKs installed to successfully run Lumberyard.
Material Editor (p. 1636)
Applies final material setup, texture mapping, and shader parameters.
Measurement System Tool
Measures the length of segmented objects like roads, rivers, and paths.
Missing Asset Resolver
Searches for assets that have moved and references their new locations (legacy).
Gems System (p. 1064)
Provides a library of prebuilt features that you can use to quickly start new projects or prototype ideas.
Particle Editor (p. 1504)
Creates and simulates explosions, fire, sparks, and other visual effects.
Project Configurator (p. 43)
Standalone application used to tell the Waf build system which Gems to include in the game build.
Resource Compiler (p. 248)
Compresses and processes source game asset files and creates package files.
Script Canvas (p. 2420)
Create game logic and behaviors with Lumberyard's new visual scripting environment.
Script Terminal
Runs various scripts in a terminal window.
Smart Objects Editor
Creates and manages smart objects, which can interact with other objects according to complex rules.
Substance Editor
Imports substance .sbsar files, edits material properties, and exports them as textures.
Terrain Editor (p. 1220)
Generates terrain and sculpts terrain elements in your level.
Terrain Texture Layers (p. 1232)
Creates and paints terrain texture layers in your level.
Time of Day Editor (p. 1265)
Creates and manages day–night cycles and other dynamic sky effects.
Track View Editor (p. 1584)
Creates and manages cinematic scenes and sequences; consists of the Track View and Curves Editor.
UI Editor (p. 2874)
Creates, manages, and simulates user interface elements for your game, such as menus and heads-up displays (HUD).
Universal Remote Console (p. 3209)
Used to connect to a remote instance of Lumberyard running on mobile devices.
Lumberyard Asset File Types

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

See the following tables for supported asset data file types in Lumberyard.

Lumberyard supports the .xml file format and the following image file formats:

- .bmp
- .jpg
- .pgm
- .png
- .raw
- .r16
- .tga
- .tif

3D Art Asset File Types

The following file formats are used for static geometry:

<table>
<thead>
<tr>
<th>File Type</th>
<th>Where Created</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.cgf (Static Geometry File)</td>
<td>DCC tool</td>
<td>Contains static geometry data, such as grouped triangles, tangent spaces, vertex colors, physics data, and spherical harmonics data.</td>
</tr>
<tr>
<td>*.chr (Character Asset File)</td>
<td>DCC tool</td>
<td>The base character used for animations.</td>
</tr>
<tr>
<td>*.cdf (Character Definition File)</td>
<td>Lumberyard</td>
<td>Defines the base character and associated attachments. This file is created with Geppetto and contains a reference to the .chr file.</td>
</tr>
<tr>
<td>*skin (Character Skinned Render Mesh)</td>
<td>DCC tool</td>
<td>Contains skinned character data, including the mesh, vertex weighting, vertex colors, and morph targets.</td>
</tr>
<tr>
<td>*.fbx (Filmbox File)</td>
<td>DCC Tool</td>
<td>Contains mesh, material, camera, and animation data. Provides interoperability between DCC tools.</td>
</tr>
<tr>
<td>*.scenesettings (Scene Settings File)</td>
<td>Lumberyard</td>
<td>Contains configuration and rules settings from an *.fbx file.</td>
</tr>
<tr>
<td>*.abc (Alembic Cache File)</td>
<td>DCC tool</td>
<td>Contains non-procedural, application-independent set of baked geometric data such as baked meshes and their materials.</td>
</tr>
<tr>
<td>*.cax (CAD/CAE Exchange File)</td>
<td>Lumberyard</td>
<td>Contains compressed game assets read from the .abc file and streamed in-game on demand from disk.</td>
</tr>
</tbody>
</table>
Asset File Types

<table>
<thead>
<tr>
<th>File Type</th>
<th>Where Created</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.trb (Terrain Block File)</td>
<td>Lumberyard</td>
<td>Contains terrain data and associated level objects such as water and vegetation.</td>
</tr>
</tbody>
</table>

Material and Texture File Types

The following files are used for the Material Editor. For more information, see Working with shaders and materials (p. 1636).

<table>
<thead>
<tr>
<th>File Type</th>
<th>Where Created</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.mtl (Material File)</td>
<td>DCC Tool</td>
<td>Contains settings for shaders, surface types, and references to textures.</td>
</tr>
<tr>
<td>*.dds (DirectDraw Surface)</td>
<td>DCC tool</td>
<td>Contains compressed source texture files.</td>
</tr>
<tr>
<td>*.sbsar (Substance Files)</td>
<td>Allegorithmic Substance Designer</td>
<td>Contains procedural materials.</td>
</tr>
</tbody>
</table>

Animation File Types

The following file types are used for the Animation Editor. For more information, see Animation Editor File Types (p. 1352).

<table>
<thead>
<tr>
<th>File Type</th>
<th>Where Created</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.actor (Actor File)</td>
<td>DCC tool</td>
<td>A character with at least one bone.</td>
</tr>
<tr>
<td>*.motion (Motion File)</td>
<td>DCC tool</td>
<td>Individual animation clips for a character, such as walk, run, and so on.</td>
</tr>
<tr>
<td>*.motionset (Motion Set File)</td>
<td>Lumberyard</td>
<td>Contains a list of motion files for a character.</td>
</tr>
<tr>
<td>*.animgraph (Animation Graph File)</td>
<td>Lumberyard</td>
<td>Contains the state machines, transitions, conditions, blend trees, and so on.</td>
</tr>
<tr>
<td>*.assetinfo (Asset Info File)</td>
<td>Lumberyard</td>
<td>Contains the configuration and settings for the *.actor and *.motion files. Animation Editor and the FBX Settings tool can create this file.</td>
</tr>
</tbody>
</table>

Audio Asset File Types

The following file types are used for the audio system. For more information, see Adding Audio and Sound Effects (p. 3140).

<table>
<thead>
<tr>
<th>File Type</th>
<th>Where Created</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.bnk (Soundbank File)</td>
<td>Audiokinetic Wwise</td>
<td>Contains compiled sound data and metadata.</td>
</tr>
</tbody>
</table>
**Lumberyard Blog, Forums, and Feedback**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

As we continue to improve Lumberyard, we want to thank everyone in our game developer community. Without your participation in the forums, your messages, and your bug reports, Lumberyard wouldn’t be as strong as it is.

- Keep sending your feedback to <lumberyard-feedback@amazon.com>.
- You can also keep up with new changes on our blog and leave comments to let us know what you think.
- Learn how to use Lumberyard our Getting Started video series. This series covers all the basics of working with Lumberyard and is a terrific foundation for moving on to intermediate and advanced topics.

**Getting Started with Amazon Lumberyard**

- Lumberyard also has an ever-growing YouTube channel that’s loaded with additional tutorial videos and presentations. Subscribe and enable notifications so you always know when new videos are released.

**Amazon Lumberyard on YouTube**

- Dive deeper into the features and usage of Lumberyard with the Amazon Lumberyard documentation.

**Amazon Lumberyard documentation home**

- Join the online community in the Lumberyard forums to learn from Lumberyard staff and users, and share your projects.

**Amazon Lumberyard forums**

- Lumberyard users have created an unofficial Discord. Join the conversation to share tips and get help from Lumberyard users.

**Amazon Lumberyard on Discord**

---

**Key Lumberyard Terminology**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you are new to working with Lumberyard, see the following list to learn about key terms and concepts for Lumberyard. For more information, see the additional topics for in-depth descriptions and procedures.
**Actor**

An actor is an entity that you create for characters in your game. You can then add the Anim Graph, Character Physics, Lua Script, and Actor components to the entity to control your actor’s movements. An Actor component exposes nodes that Lumberyard uses for animation. You work with the nodes in the Animation Editor to create motion sets that determine the motions for the characters (run, walk, idle, and so on).

**Example Actor**

The entity in the Lumberyard Editor viewport has an Actor component attached.

For more information, see Create and animate characters (p. 1340).

**Asset Builder API**

The Asset Builder API is an interface for creating custom types of assets. With the Asset Builder API, you create a custom asset, and register the asset in the asset pipeline. Asset Processor processes the asset like any other file (for example, .tif file to a .dds file), so that you can use it in Lumberyard. The Asset Builder API lets you leverage all the benefits of the asset pipeline, such as Asset Processor and the Asset Browser.

For more information, see Creating a Custom Asset Builder (p. 318).
**Asset Processor**

This utility is a background service that monitors changes to your source files and updates the cache with any changes. For example, if you replace an existing mesh asset with a new version of the asset, Asset Processor detects and processes the change for you.

**Example Importing Assets**

When you import assets into your game project, Asset Processor automatically detects the change and processes them.

For more information, see [Working with the Asset Pipeline and asset files](p. 247).

**AZ Code Generator**

This command line utility accepts a tagged source code file and creates a text, data, or code file. AZ Code Generator parses through a list of existing C++ source files and/or header files and generates the intermediate data in JSON format. It passes the intermediate data to a series of templates, which provide the format for the code that is generated.

For example, you can use the AZ Code Generator feature to create a component that you can use in the **Entity Inspector**.

**Example AZ Code Generator Process**

For more information, see [Automating boilerplate with AZ Code Generator](p. 1869).
Behavior Context

You can use the Behavior Context (p. 1053) to reflect runtime code to make it accessible to scripting languages like Lua or scripting environments like Lumberyard's Script Canvas editor. The behavior context provides bindings that invoke runtime C++ code.

Dedicated Server Build

This Lumberyard build option is a minimal Waf build of your Lumberyard project for gameplay only. This configuration can be deployed to act as a server locally or hosted on AWS. Your Lumberyard dedicated server can use the Amazon GameLift to manage your multiplayer queues.

For more information, see Creating Dedicated Servers (p. 2076).

Entity

An entity is a game object with a unique ID and a container. An entity contains one or more components that provide functionality. For example, you can add a Mesh component to an entity to add visual geometry, such as creating a door or tree for your game.

Example Entities and Components

You can see entities in the Entity Outliner and the components attached to that entity in the Entity Inspector.

For more information about entities and components, see Working with component entities (p. 462).

Event Bus System

Event buses (EBuses) are a general-purpose communication system that Lumberyard uses to dispatch notifications and receive requests. EBuses are configurable and support many different use cases. For example, you can use EBuses to notify other parts of the engine about an event, such as changes to a component's configuration.

You can use the event bus system by adding a component to an entity, as a call in C++, or by including the API header.

For more information, see Working with the Event Bus (EBus) system (p. 1851).
GameLift

Amazon GameLift is an AWS service that you can use to host server-based games, scale servers up and down, allow players to connect, and provide matchmaking services. You can enable the GameLift gem for Lumberyard.

For more information, see Using Amazon GameLift (p. 2088) and GameLift Gem (p. 1130).

Gem

Gems are reusable packages of code and assets that can be added, modified, or removed from a Lumberyard game. A gem is a module based on the AZ Module system (p. 1090). You can enable gems for your game for additional assets and code. For example, you can enable the Rain gem so that you can add the Rain and Snow components to your entities. Gems can be enabled in other game projects or distributed to other game developers. Lumberyard builds gems with the Waf build system. Some gems are required for Lumberyard.

Cloud gems are Lumberyard gems that provide AWS cloud-connected functionality for a game. For more information, see Cloud Gems (p. 2127).

For more information, see Creating a Gem (p. 1064).

Example Gems

You can enable gems for your game project with the Project Configurator.

GridMate Networking System

GridMate is the Lumberyard networking library for hosting multiplayer gameplay on client-server or peer-to-peer architectures. You can enable the Multiplayer gem to include this library and the NetBindable interface for your project.

For more information, see Using Lumberyard Networking (p. 1990) and Multiplayer Gem (p. 1171).

Slice

A slice is a collection of configured entities that are stored as a single unit in a reusable asset. You can use slices to group entities and other slices for reuse. With slices, you can edit, create, and spawn
multiple entities as a set. Your project can use a slice in a similar way that you used prefabs and layers in previous versions of Lumberyard.

Slices contain entities, including their components and properties, and may also contain instances of other slices. This ability to nest slices without flattening the hierarchy is a feature unique to Lumberyard.

**Example Slice**

*Simple_JackLocomotion* is a slice that contains two entities – *Jack* and *Camera*. You can find this slice in the Samples Project (p. 146).

For more information, see Working with Slices (p. 510).

**Virtual File System (VFS)**

This system is designed for non-PC platforms to handle game assets for live reloading. The virtual file system streams and loads your assets from your development machine as needed. You do not have to load your entire level or game to the device ahead of time and you can avoid deploying assets to a target device. When you make a change on your development machine, live reloading automatically updates the asset in Lumberyard Editor and for the target device.

For example, if you want to run your game on an iOS device and you change an asset, Asset Processor uses the virtual file system to stream the updated asset from your development machine to the iOS device.

For more information, see Live Reloading and VFS (p. 291) and Serving assets over the Virtual File System (VFS) (p. 3174).
Working with Lumberyard projects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Working with projects in Lumberyard

Learn about the details of creating, configuring, managing, and building your Lumberyard game projects, along with the sample projects provided in your installation.

In this section:

<table>
<thead>
<tr>
<th>Topic Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a project (p. 43)</td>
<td>Learn how to create and configure a new Lumberyard game project with Waf.</td>
</tr>
<tr>
<td>Manage a project with Lmbr.exe (p. 56)</td>
<td>Learn how to manage the configuration and details of your Lumberyard game project.</td>
</tr>
<tr>
<td>Build a project (p. 61)</td>
<td>Learn how to build your Lumberyard game project with Waf.</td>
</tr>
<tr>
<td>Sample projects and levels (p. 142)</td>
<td>Learn about the sample game projects provided in the Lumberyard installation, and discover examples that can accelerate your understanding of Lumberyard and your game's development.</td>
</tr>
</tbody>
</table>

Creating Lumberyard projects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Project Configurator is a standalone application that allows you to specify to the Waf build system which game projects and assets (Gems (p. 1064)) to include in a game build. With it, you can create new projects, save active projects, configure advanced settings, and enable, disable, or create new Gems (p. 1064). You can also use Lmbr.exe (p. 56) to perform these functions in a command line environment.

For information about Waf build system, see Using the Waf Build System (p. 63). For information about Gems, see Add modular features and assets with Gems (p. 1064).

Note
Before you can run the Project Configurator, you must first run Lumberyard Setup Assistant (p. 16) and close Lumberyard Editor.
Creating and Switching Game Projects

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Project Configurator to create and change the settings that affect building and editing of a game project. You can use the Project Configurator to do the following:

- Create a game project
- Enable or disable gems (code and assets files) from your game project
- Switch game projects

To learn about creating and compiling a new game project for Lumberyard, watch the following video tutorial.

How to Create and Compile a New Project for Amazon Lumberyard

Topics

- Project Templates (p. 44)
- Creating a Game Project in Lumberyard (p. 45)
- Choosing a Game Project to Open (p. 47)
- Switching Game Projects (p. 48)

Project Templates

When you create your game project with the Project Configurator, you can use the following templates:
Empty Template

The Empty template has the minimum features required for the editor to load and run a game project:

- **CryLegacy** – Enables the editor and launcher to load a game project that contains legacy code
- **Legacy Game Interface** – Enables the editor and launcher to load a game project that contains legacy game interface code
- **LyShine gem** – Provides access to the in-game UI system for Lumberyard
- **Maestro gem** – Provides access to cinematics features

Default Template

The Default template builds on the Empty template and enables the following gems to provide basic features for game development:

- **Amazon GameLift** – Provides capabilities to use this service and create game sessions
- **Camera** – Includes a basic camera component for runtime rendering
- **ChatPlay** – Includes the interface for triggering events based on Twitch chat activity
- **Cloud Canvas** – Provides visual scripting capabilities to power your game backend with AWS services
- **Gestures** – Allows gesture-based input, including click/tap, drag, hold, pinch, rotate, and swipe
- **Http Requestor** – Adds support to handle HTTP and HTTPS requests
- **In-App Purchases** – Provides the in-app purchasing API for Android and iOS
- **Input Management Framework** – Converts input to user-defined gameplay events
- **LyShine** – Provides access to the in-game UI system for Lumberyard
- **PBS Reference Materials** – Includes a set of physically based shading reference materials and texture assets
- **Physics Entities** (Legacy) – Provides physics entity modifiers to simulate physical events
- **Primitive Assets** – Provides primitive objects to manipulate in your level

To see the gems that enabled for the Default template, create a project from that template and choose Enable Gems.

The Default template also includes a simple level for you to manipulate components and entities in the editor and experiment with other assets in a neutral, gray environment. The simple level includes a camera, a single light, an environment probe for reflections, and primitive objects with physics enabled. The objects are provided by the Primitive Assets Gem (p. 1190).

**Note**

- When you create your game project, we recommend the Default template and its set of gems. This template helps you start designing and developing quickly. Some of the gems in the template are optional.
- To create a base project without optional gems, use the Empty template. This template creates a basic project using the minimum features to start a game.

Creating a Game Project in Lumberyard

You can create a game project using the Project Configurator or command line. When you create a game project, the following are also created:

- A game gem that includes your game-specific code. The gem is named after your game project and is located in the MyNewProject\Gem directory.
• A Visual Studio solution file specific to your game.

**Project Configurator**

**To create a game project in the Project Configurator**

1. Open the Project Configurator with your preferred method:
   - Open Lumberyard Setup Assistant and, on the **Summary** page, click **Configure project**
   - Open the Lumberyard Project Configurator, located at `lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe`. For example, when using Visual Studio 2019 as your build platform, the Project Configurator is located at `lumberyard-version\dev\Bin64vc142\ProjectConfigurator.exe`.

2. In the Project Configurator, click **Create new**.

3. In the **Create a new project** window, do the following:
   a. Enter a name for your project.
      Only alphanumeric characters are allowed. Don't use special characters or spaces in the name. Don't use single C++ keywords such as if, while, break, new, and virtual.
   b. Choose one of the following: **Default** or **Empty**.
   c. Click **Create project**.

4. When your project is created, click **Continue**.

5. In the Project Configurator, select the new project and click **Set as default** to make it the default project that Lumberyard Editor loads.

6. **(Optional)** Click **Enable Gems** to add gems to your project.

   **Note**
   If you selected **Code & Assets** gems, you must build your project.
   a. At a command-line prompt, navigate to the `lumberyard_version\dev\` directory.
   b. To configure Lumberyard, enter the following command.

   ```
   lmbr_waf configure
   ```
   c. Build the game project.
Note
In Lumberyard version 1.23, the initial build time for a new project can be lengthy. It can take approximately one hour on higher end systems.

For more information, see Building Lumberyard projects (p. 61).

7. Close the Project Configurator.
8. Open Lumberyard Editor with your preferred method:
   • Open Lumberyard Setup Assistant and, on the Summary page, click Launch editor
   • Start Editor.exe from one of the following directories:
     • For Visual Studio 2017: lumberyard_version\dev\Bin64vc141
     • For Visual Studio 2019: lumberyard_version\dev\Bin64vc142

Command Line
When you create a game project from a command line, you can enter lmbr for a list of all possible commands that you can use with Lmbr.exe (p. 56).

To create a game project from a command line
1. At a command-line prompt, navigate to the following directory:

   lumberyard_version\dev\Tools\LmbrSetup\Win\n
2. To create your project, enter the following command.

   lmbr projects create MyNewProject

3. To set your new project as the default or active project, enter the following command.

   lmbr projects set-active MyNewProject

4. To configure Lumberyard to run this project, navigate to the following directory:

   lumberyard_version\dev\n
5. Enter the following command.

   lmbr_waf configure

6. Build the game project.

   For more information, see Building Lumberyard projects (p. 61).

Choosing a Game Project to Open
Use the Project Configurator to set default game project that opens in Lumberyard Editor.

To open a game project
1. Open the Project Configurator with your preferred method:
   • From the desktop, double-click the Project Configurator icon
   • Open the Lumberyard Project Configurator, located at lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe. For example, when using Visual Studio 2017 as your build platform, the Project Configurator is located at lumberyard-version\dev\Bin64vc141\ProjectConfigurator.exe.
Creating and Switching Game Projects

2. On the Summary page, select the project and click Set as default.

3. (Optional) Click Enable Gems and select the additional features and assets that you want available to design your game. If you select a gem labeled with Code & Assets, you must build your project from a command line.

   For more information, see Building Lumberyard projects (p. 61).

4. (Optional) Choose Advanced Settings and verify that the System entity and Memory settings are appropriate for your project.

5. Close the Project Configurator.

6. After setting the default project, open Lumberyard Editor with your preferred method:

   • From the desktop, double-click the Lumberyard Editor icon
   • For Visual Studio 2017, navigate to the \lumberyard_version\dev\Bin64\vc141 directory and double-click Editor.exe
   • For Visual Studio 2019, navigate to the \lumberyard_version\dev\Bin64\vc142 directory and double-click Editor.exe

   After the first launch and each time you change projects, Asset Processor runs in the background. Asset Processor manages your assets and project files. While Lumberyard Editor opens, you can see messages appear with status information.

   For more information, see Using Asset Processor (p. 248).

Switching Game Projects

To switch projects, you must close Lumberyard Editor. With the Project Configurator, you then specify a different project and its gems and system settings for the editor to open.

When you switch projects, you can specify the following:

• Name of the project to open
• (Optional) Gems enabled for the project
• (Optional) System settings for the editor to use

To switch game projects

1. In Lumberyard Editor, choose File, Project Settings, Switch Projects.
2. You must close Lumberyard Editor before you can open the Project Configurator. In the dialog box, choose Save.

3. Follow the procedure in Choosing a Game Project to Open (p. 47).

Configuring Advanced Settings

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To configure advanced settings for your selected project, click Advanced Settings in the Project Configurator.

Upon launch, the System Entity Editor detects any required system components that are missing and adds them. System Entity Editor displays a dialog that lists all of the components that it added to your project.
The System Entity Editor interface has the following tabs:

- System Entity
- Memory Settings

**System Entity Editor**

A single entity, shown in the following picture as SystemEntity, makes up the core of every Lumberyard application. This entity’s components, known as system components (p. 1098), power major systems within Lumberyard. Using the System Entity Editor, game developers can customize the project’s components and configuration settings.
To add a system component, click **Add Component**.

**Memory Settings**

Click the **Memory Settings** tab to configure and manage your project's memory settings.
Understanding the Project Configurator

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Project Configurator is an application for specifying the settings that affect building and editing of your game project. When you work with the Project Configurator, note the following changes.

Topics
Set Default (Active) Project

When you set the default (active) project, the Project Configurator modifies the following files with the following information:

```
lumberyard_version\dev\bootstrap.cfg
```

The `sys_game_folder` property specifies the project that the editor and PC launcher attempts to load (for example, `sys_game_folder=SamplesProject`).

```
lumberyard_version\dev\WAF\user_settings.options
```

The property `enabled_game_projects` is a comma-separated list of one or more projects to include in a build.

Create a New Project

When you create a new project in the Project Configurator, several things happen:

- The Project Configurator uses the EmptyTemplate to create a new project. This template is located in the `lumberyard_version\dev\ProjectTemplates\EmptyTemplate` directory.
- The contents of `lumberyard_version\dev\ProjectTemplates\EmptyTemplate\code\EmptyTemplate` are copied to the following directory:
  
  `lumberyard_version\dev\code\NewProjectName`

- The contents of `lumberyard_version\dev\ProjectTemplates\EmptyTemplate` are copied to the following directory:
  
  `lumberyard_version\dev\NewProjectName`

- The EmptyTemplate is replaced in both new directories with the name of your new project, including file names and file contents.

For more information about templates, see Project Templates (p. 44).

Enable or Disable Gems

When you enable or disable a gem in a project, the Project Configurator updates the list of enabled gems that is maintained in `lumberyard_version\dev\Game_Project\gems.json`.

For more information, see Enabling Gems (p. 1064).

- Enabling a gem adds that gem name to the list in `gems.json`.
- Disabling a gem removes that gem name from the list in `gems.json`.
- Enabling or disabling a gem updates the `Editor.xml` and `Game.xml` files. You can find these files in the `lumberyard_version\dev\Game_Project\Config` directory. These files tell Lumberyard to load the required `.dll` files for the gems that you specify.
**Note**
The Editor.xml and Game.xml files are only updated when you enable or disable gems where "LinkType": "Dynamic" in the gem.json file. For more information, see Gem JSON File (p. 1074).

**Create a New Gem**

When you use the Project Configurator to create a gem, the following takes place:

- The content of the gem template (which is part of the Project Configurator resources and is not directly accessible) is copied into the `lumberyard_version\dev\Gems\Gem_Name` directory.
- Strings with the gem name in the copied content are replaced with the name of your gem to make it a valid gem.

For more information, see Creating a Gem (p. 1066).

**Troubleshooting**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Review the following if you experience issues when using the Project Configurator.

**Perforce integration**

If you have enabled Perforce integration with the tooling, then the Project Configurator automatically checks the following files out of Perforce. If not, then update the read only flag manually:

- `project_asset_folder\gems.json`
- `project_asset_folder\game.cfg`
- `lumberyard_version\bootstrap.cfg`
- `lumberyard_version\dev\game_project_folder\project.json`

**Cannot create a new project**

Make sure that the `lumberyard_version\dev\game_project_folder\project.json` file is editable.

Ensure that the name entered is valid and does not contain special characters or whitespaces.

**Cannot enable or disable a gem**

Make sure that the `project_asset_folder\gems.json` file is editable before trying to save changes made to gems being enabled or disabled.

**New project or gem does not appear in Visual Studio**

Make sure that you have run `lmbr_waf configure` from a command line, which regenerates the Visual Studio solution to include the new project or gem.
If the project or gem still does not show up in Visual Studio, ensure that the `enabled_game_projects` field in the `lumberyard_version\dev\_WAF\_\user_settings.options` file is set to the name of your project.

**Wrong project gets loaded in Lumberyard Editor**

Ensure that the `lumberyard_version\dev\bootstrap.cfg` is editable. Then, open the Project Configurator, select the project to open, and choose **Set as default**.

Also ensure that the `sys_game_folder` field in the `lumberyard_version\dev\bootstrap.cfg` file is set to the name of your project.

**Managing Game Projects with Lmbr.exe**

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/gameengine/) or visit the [AWS Game Tech blog](https://aws.amazon.com/developer/game-tech/) to learn more.
Lmbr.exe is a command-line tool for managing capabilities, game projects, and Gems.

To find the Lmbr.exe file

1. Navigate to one of the following directories:
   - For Visual Studio 2017: `lumberyard_version\dev\Bin64\vc141`
   - For Visual Studio 2019: `lumberyard_version\dev\Bin64\vc142`
2. In a command line prompt, enter the following command to see the commands that you can run.
   
   ```
   lmb -help
   ```

   The list of possible commands appears.

   ![Available Commands](image)

   You can also use `-help` on other commands. For example, to see more information about Lumberyard capabilities, you can enter the following commands:

   ```
   lmb capabilities -help
   lmb capabilities list -help
   ```

   Topics
   - Projects Commands (p. 58)
   - Gems Commands (p. 58)
   - Capabilities Commands (p. 59)
   - ThirdPartySDKs Commands (p. 60)
   - Packages Commands (p. 60)
Projects Commands

Use the following commands to create and modify game projects.

**list**

Lists all projects in the current engine directory.

```
lmbr projects list
```

**create**

Creates a new project using EmptyTemplate, which is located at `dev\ProjectTemplates\EmptyTemplate`, as a template.

```
lmbr projects create project_name
```

**get-active**

Displays the name of the active project.

```
lmbr projects get-active
```

**set-active**

Sets the active project for building and executing Lumberyard. This command modifies `_WAF_\user_settings.options` and `bootstrap.cfg` to reference the specified project.

```
lmbr projects set-active project_name
```

**populate-appdescriptors**

Populates the appdescriptors files from the gems list. If you use the Project Configurator to change gems, Lumberyard automatically updates the application descriptor files. If you manually edit a `gems.json` file, run this command to update these two application descriptor files in a project's asset directory:

- `dev/project_asset_directory/Config/Game.xml`
- `dev/project_asset_directory/Config/Editor.xml`

```
lmbr projects populate-appdescriptors
```

Gems Commands

Use the following commands to create gems and modify a project's use of gems.

**list**

Lists all gems that are installed or enabled in the specified project.

```
lmbr gems list (-project project_name)
```
disable

Disables the specified gem in the specified project. If -disable-deps is specified, all dependencies of the gem are also disabled.

```
lmbr gems disable project_name gem_name (-disable-deps)
```

enable

Enables the specified gem in the specified project. If a version is specified, it's used, otherwise the latest version installed is used.

```
lmbr gems enable project_name gem_name (-version version)
```

create

Creates a gem with the given name. If you specify id or version, those values are used. If you do not specify -out-folder, name is used.

```
lmbr gems create gem_name (-version version) (-out-folder gems\relative_folder)
```

Capabilities Commands

Use the following commands to create and modify Lumberyard capabilities.

list

Lists all of the Lumberyard capabilities.

```
lmbr capabilities list
```

create

Creates a new Lumberyard capability. Include the ID, description, and tooltip.

```
lmbr capabilities create new_capability_name "This is the description." "This is the tooltip."
```

You can also specify the following arguments:

- -help (bool): Displays help descriptions of available commands and options.
- -tag (list): String tags associated with the capability.
- -default (bool): True if capability is default; otherwise false.
- -enable (bool): True if you want to enable the capability right away.

disable

Disables a Lumberyard capability.

```
lmbr capabilities disable lumberyard_capability
```

enable

Enables a Lumberyard capability.
**ThirdPartySDKs Commands**

Before you can use `thirdpartysdk` commands, you must set up your workspace. To do this, run Lumberyard Setup Assistant and perform each step of the setup process.

The `thirdpartysdk` module manages the installation states of third party SDKs. These are specified by default in the `\3rdParty\` directory. If you have defined a different third party SDKs path in the `SetupAssistantUserPreferences.ini` configuration file, then this module manages the SDKs in that path.

The `thirdpartysdk` module has dependencies on the following:

- **Engines module**
- **Capabilities (p. 59) module** – Downloads only the SDKs that are required based on the user-enabled capabilities.
- **Packages (p. 60) module**
- `SetupAssistantUserPreferences.ini` – Path specified for the SDKSearch3rdParty property.
- `SetupAssistantConfig.json` – Parses the SDK required for the engine

Use the following commands to manage Lumberyard third party SDKs.

**list**

Prints out a list of SDKs specified by `SetupAssistantConfig.json` and their available state.

```
lmbr thirdpartysdk list (-help)
```

**setup**

Downloads and sets up all Lumberyard third party SDKs based on your current selected capabilities (p. 59). Does not support software installation such as FBX or Android SDK.

```
lmbr thirdpartysdk setup (-only <sdk1,sdk2,sdk3>) (-3rdpartypath <path>) (-help)
```

- `- (only) (list): Specify name(s) of specific SDKs to set up (comma separated).
- `- (3rdpartypath) (string): Path to your third party folder. Defaults to path in `SetupAssistantUserPreferences.ini`.
- `- (optionals) (flag): Flag to include optional sdks. Does not work if specifying specific sdks.
- `- (help): Print help describing available commands/options.

**Packages Commands**

Before you can use `packages` commands, you must set up your workspace. To do this, run Lumberyard Setup Assistant and perform each step of the setup process.
The packages module downloads third party packages from S3.

The packages module has dependencies on the Engines module.

Use the following command to download a resource packages from S3.

```
dlmb packages download [name] [version] [platform] [uri] [destination] (-help)
```

- `[name]` (positional argument): The name of the package you want to download.
- `[version]` (positional argument): the version of the package you want to download.
- `[platform]` (positional argument): The platform of the package you want to download.
- `[uri]` (positional argument): Where to look for package on remote file system.
- `[destination]` (positional argument): Where to unpack the package.
- `(-help)`: Print help describing available commands/options.

Building Lumberyard projects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Building your Amazon Lumberyard game is the most important part of the development process - it's how you debug, test, and distribute. In this section of the guide, you'll learn how to configure your Lumberyard builds, get started with your first build, see which build types are available and what they offer, and how to use the Waf build system.

For instructions on bundling assets and distributing them with a release build, see Build and bundle assets for release in Lumberyard (p. 344).

**Your First Lumberyard Build**

One of the first things you'll want to do with Lumberyard is configure the build system and see your configured game project (p. 43) running as a stand-alone executable or Visual Studio Solution.

**Configure and build a Lumberyard game**

1. Open a command line window and navigate to the `Lumberyard_version\dev` directory.
2. Configure the build system:

   ```
   lmbr_waf configure
   ```

3. Build your game project in profiling mode:

   ```
   lmbr_waf build_win_x64_vs2019_profile -p game_and_engine
   ```

   Depending on your hardware, the build may take a while to complete.

To learn the general steps for creating a release build in Lumberyard, see Build and bundle assets for release in Lumberyard (p. 344). For more information about creating release builds for Android and iOS, see Developing for Android and iOS with Lumberyard (p. 3166).
### Topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game Build Types in Amazon Lumberyard (p. 62)</td>
<td>Learn about the available types of Amazon Lumberyard builds and when to use them during development.</td>
</tr>
<tr>
<td>Using the Waf Build System (p. 63)</td>
<td>Learn about the Waf build system, which is used by Lumberyard to configure and build game code.</td>
</tr>
<tr>
<td>Build and bundle assets for release in Lumberyard (p. 344)</td>
<td>Run through a tutorial and learn how to use the Asset Bundler, along with Waf, to create release builds.</td>
</tr>
<tr>
<td>Adding Custom Game Icons (p. 141)</td>
<td>Add a custom icon to display in your game's window as part of a release.</td>
</tr>
</tbody>
</table>

### Game Build Types in Amazon Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Amazon Lumberyard lets you create different types of game builds for each step in your development process:

**Profile mode builds for game developers, designers, and artists**

- Provides an optimized build meant for game development.
- Contains performance instrumentation and debugging output.
- Compiles shaders at run time, which may require the remote shader compiler.
- Communicates with Asset Processor and compiles as needed.
- Has logging, crash reporting, metrics, and other troubleshooting features.

**Debug mode builds for programmers**

- Provides a non-optimized version of the game engine that provides the same features as the profile version.
- Has additional memory checks and tests.
- Provides step-by-step code that programmers can use to debug the execution.

**Release mode builds for customer previews, demos, and launches**

- Loads assets and data only from .pak files.
- Loads shaders from .pak files for better performance but may compile shaders for DirectX at run time if a shader is not found in the .pak files.
- Can't use virtual file system (VFS) or remote asset access.
- Doesn't communicate with Asset Processor because Asset Processor doesn't ship with the game.
- Removes most logging, instrumentation, profiling, and other measurement metrics.
• Removes all game developer and programmer features such as console usage, cheat commands, command line parsing, and batch mode processing.
• Combines everything into a single executable file instead of DLLs.

To learn how to perform each type of build and about build configuration, see Waf Commands and Options (p. 90)

Using the Waf Build System

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard uses the Waf build system to allow you to switch between various build pipelines and to ensure you build only what is needed. You can use extensions, such as automatic project generation, or a simple GUI to modify the command line base system for your project requirements.

You can run Waf using the following methods:

• Command line window

• Waf-generated, Visual Studio solution file. Waf creates a Visual Studio solution file along with the projects specified in the selected project specs. If more than one spec file includes the same project, only one project file is created to prevent duplicates. Waf uses the project specs to determine the projects, project filters, and possible build configurations. Waf uses the wscript files to identify individual project definitions.

Note
Waf is compatible with the version of Python 3 that comes bundled with Lumberyard.

Lumberyard includes the Project Configurator (p. 43), a standalone application that allows you to specify to Waf which game templates and assets (Gems) to include in the game build.

Topics
• Waf File System (p. 64)
• Waf Commands and Options (p. 90)
• Waf Supported Operating Systems and Compilers (p. 95)
• Waf Project Settings (p. 96)
• Using the Waf Artifacts Cache (p. 99)
• Adding Third-Party Libraries (p. 105)
• Creating Third-Party Library Configuration Files for Waf (p. 108)
• Waf Extensions (p. 118)
• Using Waf (p. 120)
• Adding User Settings to Waf (p. 133)
• Adding Qt 5 Content to Waf (p. 136)
• Using Uber Files (p. 139)
• Debugging Waf (p. 140)
Waf File System

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can find global configurations and project specs in the _WAF_ directory at the root project path. Three subfolders represent settings specific to the following build systems: android, apple, and msbuild. Defined specs are located in the specs directory.

In addition to the configurations specified in the _WAF_ directory, you can find other Waf settings in the waf_branch_spec.py file in the root directory. You can modify this file if you need to include support for additional platforms or configurations.

The Waf build file system can be grouped into three categories:

- Waf Module files (wscript)
- Waf file list (*.waf_files)
- Project and compilation files such as *.h, *.cpp, and so on

Topics

- Waf File Lists (*.waf_files) (p. 64)
- Waf Branch Spec (waf_branch_spec.py) (p. 67)
- Waf Projects File (project.json) (p. 69)
- Waf Spec Files (*.json) (p. 74)
- Waf Module Files (wscript) (p. 76)
- Waf User Options and Settings (p. 78)

Waf File Lists (*.waf_files)

In standard Waf, the source files to include for a build is specified directly as a list of files using the keyword Source. However, the list of source files does not provide support for the following features in Lumberyard:

- **Uber Files**

  Uber files provide a way to group files into a single compilable unit. This grouping offers the benefits of improving build speed and creating smaller binaries. However, it is not always possible to combine all files together successfully without changing the source code. The .waf_files file list provides a way to manually group files and avoid potential collisions.

- **Visual Studio Filters**

  The .waf_files mechanism also provides a way to declare the source files that are included in a Microsoft Visual Studio filter. Visual Studio's solution explorer does not always accurately represent the files and their directory hierarchy on disk, so it uses filters. Cases exist in Lumberyard where multiple projects share a subset of source code outside of individual project directories. The ability to manually specify filters lets Lumberyard maintain this file sharing.
Using the Waf Build System

The paths to the source files specified in a .waf_files file are relative to the location of the .waf_files file itself. It is also possible to use aliases (p. 113) to third-party directories for situations in which a third-party library consists of source code.

File Structure

A .waf_files file is organized into three levels:

- **Level 1 – Uber file grouping**
  
  When uber files are enabled for the build, Waf uses this dictionary of uber file name keys to generate the uber files. The uber files combine the files specified in the group. Because uber files are relevant only for C++ source, the names of the uber files must have a C++ source file extension (for example, .cpp or .cc) and must be unique within the list.

  Level 1 key names can also include the following special key words:
  
  - **none** – Files under the none grouping are not combined into any uber file, even if uber mode is enabled.
  - **auto** – Files are combined automatically into uber files based on a set uber file size limit.

- **Level 2 – Visual Studio filter specification**
  
  The key names for this dictionary represent the directory filters in the generated Visual Studio projects from Waf. The name supports a path-like definition that uses ‘/’ delimiters that you can use to specify directory structures for the filter. If the source files are intended to reside in the root level of the project in Visual Studio, the reserved key name Root is used.

- **Level 3 – List of source files**
  
  The list of source files relative to the location of the .waf_files file to be listed under the Visual Studio filter key name. Waf file lists also support file globbing. For more information, see the File Globbing section in this topic.

Using Uber Mode

Uber mode is specified in the user_settings.options file under the use_uber_files attribute, as shown in the following example:

```python
use_uber_files = False
```

When uber mode is enabled, Waf uses the Level 1 grouping information to create uber files to combine for compiling.

In deciding how to use uber mode, consider the following suggestions:

- If you do not want to create an uber file and want to compile the files individually, specify none for the group of files.
- If you want Waf to automatically place files into uber files, specify auto.
- If you want to specify an uber source file to group the source files, specify the name of that uber source file.

The following example shows this structure.

```json
{
    "none": {
        "Source Files": [
            ...
```
Limiting the Uber File Byte Size

When you specify `auto`, Waf limits the size of the uber file to the byte limit specified in the `user_settings.options` file `uber_file_size` attribute, as in the following example:

```
uber_file_size = 307200
```

File Globbing

The source file definitions support limited ant pattern-based file matching, or "globbing". This allows you to group related files together based on file extension instead of listing files individually. To enable globbing, use a globbing pattern instead of a file path in the source file. The following shows examples of file globbing.

```
{
  "none": {
    "Root": [
      "*.cpp"
    ],
    "Single": [
      "Single/*.cpp"
    ],
    "Nested": [
      "Nested/**/*.cpp"
    ]
  }
}
```

The example showcases the following globbing patterns:

<table>
<thead>
<tr>
<th>Globbing Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*.cpp</code></td>
<td>Add all files with the <code>.cpp</code> extension in the current directory only.</td>
</tr>
<tr>
<td><code>Single/*.cpp</code></td>
<td>Add all files with the <code>.cpp</code> extension in the <code>Source</code> directory only.</td>
</tr>
<tr>
<td><code>Nested/**/*.cpp</code></td>
<td>Recursively add all files with the <code>.cpp</code> extension in the <code>Source</code> directory.</td>
</tr>
</tbody>
</table>
**Custom Globbing Rules**

You can create some custom globbing rules by specifying a dictionary for the glob search. For example, the following dictionary includes all files in the directory except for files with a .res extension:

```plaintext
{
    "none": {
        "Root": [
            {
                "pattern": "**/.*",
                "excl": "*.res"
            }
        ]
    }
}
```

Dictionary specifications support the following keys:

- **pattern** – (Required) The globbing pattern to apply to the search.
- **excl** – An exclusion pattern to filter out of the results.
- **maxdepth** – For a recursive search, limit the depth of the search.
- **ignorecase** – Ignore case when using the globbing pattern.

**Globbing at Configure Time and Build Time**

Because globbing is an expensive operation, it is not recommended on large sets of files. Globbing can be done either at configure time, or at both configure and build time. Doing globbing only at configure time improves the build performance. It reuses the glob result that was created at the time of the last configure. The disadvantage is that it does not pick up any files that are added or removed until another configure is done. Doing globbing at both configure and build time picks up file additions and removals for every build, but can be expensive.

**Enabling Build Time Globbing**

To enable or disable build time globbing, set the `enable_dynamic_file_globbing` attribute in the `user_settings.options` file to `True` or `False`, as shown in the following example:

```plaintext
enable_dynamic_file_globbing = False
```

**Waf Branch Spec (waf_branch_spec.py)**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The `waf_branch_spec.py` is the topmost configuration level of the Waf build system. It specifies which operating systems and configurations are available for all projects and specs.

The following is an example `waf_branch_spec.py` file, with explanatory comments for each value moved and enhanced in the table that follows:

```plaintext
# Global constants
BINTEMP_FOLDER = 'BinTemp'
WAF_FILE_GLOB_WARNING_THRESHOLD = 1000
CACHE_FOLDER = 'Cache'
```
The `waf_branch_spec.py` file manages the following global values:

**Global values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDITIONAL_COPYRIGHT_TABLE</td>
<td>(Optional) To embed a company name and copyright in the generated binaries, add a name-value pair here. Example:</td>
</tr>
</tbody>
</table>
|                           | ADDITIONAL_COPYRIGHT_TABLE = {
|                           |     'MyCompany' : 'Copyright (c) MyCompany'
|                           | }
|                           | You must also add 'copyright_org' to either your `project.json` or `wscript` module definition file. |
|                           | Project.json example:                                                        |
|                           | "copyright_org": "MyCompany"                                                |
|                           | Wscript module example:                                                      |
|                           | bld.CryEngineModule(                                                         |
|                           |     ...                                                                    |
|                           |     copyright_org = 'MyCompany'                                              |
|                           |     ...                                                                    |
|                           | )                                                                        |
| ADDITIONAL_SEARCH_PATHS   | (Optional) Provide additional paths to search for the WAF build. You can use aliases such as @ENGINE@ and @PROJECT@ for the engine and project roots, respectively. |
| ADDITIONAL_WAF_MODULES    | (Optional) Specify a table of additional modules that will be loaded by WAF. |
|                           | The table format is:                                                        |
|                           | 'Key' : [ 'Module list' ]                                                  |
|                           | Where key is the path of the directory for a set of WAF modules, and the module list is a list of WAF python modules to load into the build system, separated by commas and relative to the path directory specified by the key. |
|                           | Example:                                                                   |
### Using the Waf Build System

**Value**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| ADDITIONAL_WAF_MODULES = {
  'Tools/Build/custom_build': [
    'custom_a.py',
    'custom_b.py:win32'
  ]
} |

The above example will load 'custom_a.py' for all platforms, and 'custom_b.py' only for win32 platforms. Note that the methods that are exposed in the modules must be decorated accordingly, as they are generally used based on the context of the command, and not through regular python imports.

**BINTEMP_FOLDER**

Subfolder under the base of the project where WAF stores all intermediate and temporary files.

**LMBR_WAF_VERSION_TAG**

Version stamp (GUID) of lmbrwaf that is used to signal that a clean of bintemp is necessary. Only update this number if there are changes in WAF handling where it is not possible to track stale intermediate files caused by the WAF changes. To ignore the bintemp cleaning check, set this value to `None`. Only update this value as a last resort. If there were WAF changes that do not affect the generation or tracking of intermediate of generated files, then there is no need to wipe out BinTemp.

**WAF_FILE_GLOB_WARNING_THRESHOLD**

Defines a warning threshold in number of files that were hit during `waf_file globbing (p. 66)`.

### Waf Projects File (project.json)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/developerguides/3d-engine/) or visit the [AWS Game Tech blog](https://aws.amazon.com/blogs/gaming/) to learn more.

The `project.json` file (located in each game project directory) is used to store game project-specific data. The `enabled_game_projects` settings (`user_settings.options`) and the `--enable-game-projects` build parameter use the project names defined in this file.

The `project.json` file is structured as follows:

- **First level** – Represents the project based on its name
- **Second level** – Presents attributes that you can set for each game project

The following is an example `project.json` file:

```json
"project_name" : "SamplesProject",
"product_name" : "Samples Project",
"executable_name" : "SamplesProjectLauncher",
"code_folder" : "Code/SamplesProject",
"modules" : ["SamplesProject"],
"project_id": "{D882E365-54D6-586E-BD78-2650F3057D49}"
```

You can configure the following settings in the `project.json` file:
### General settings

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>executable_name</td>
<td>Name of the built executable file:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Dedicated server executables</strong> – '_Server' is appended to the name</td>
</tr>
<tr>
<td></td>
<td>• <strong>Unit test executables</strong> – '_UnitTest' is appended to the name</td>
</tr>
<tr>
<td>modules</td>
<td>(List) Base modules for the game</td>
</tr>
<tr>
<td>product_name</td>
<td>Externally-facing name of the product</td>
</tr>
</tbody>
</table>

The following values are only valid under the **android_settings** key:

### Android settings

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_name</td>
<td>Android application package identifier. Used for generating the project-specific Java activity class and in AndroidManifest.xml. Must be in dot separated format.</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;com.lumberyard.sdk&quot;</td>
</tr>
<tr>
<td>version_number</td>
<td>Internal application version number. Used to set the android:versionCode tag in AndroidManifest.xml.</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td>version_name</td>
<td>Human readable version number. Used to set the android:versionName tag in AndroidManifest.xml.</td>
</tr>
<tr>
<td></td>
<td>Example: &quot;1.2.3-beta&quot;</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;1.0.0.0&quot;</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;landscape&quot;</td>
</tr>
<tr>
<td>icons</td>
<td>A map of icon override path(s) for each screen DPI type. All entries require a path that is either relative to &lt;engine&gt;\Code&lt;project&gt;\Resources or an absolute resource path to the PNG image.</td>
</tr>
<tr>
<td></td>
<td>Available sub-options:</td>
</tr>
<tr>
<td></td>
<td>• default</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>• Default image used if a specific DPI is not specified</td>
<td></td>
</tr>
<tr>
<td>• mdpi</td>
<td>Medium ~160 dpi</td>
</tr>
<tr>
<td>• hdpi</td>
<td>High ~240 dpi</td>
</tr>
<tr>
<td>• xhdpi</td>
<td>Extra high ~320 dpi</td>
</tr>
<tr>
<td>• xxhdpi</td>
<td>Extra-extra high ~480 dpi</td>
</tr>
<tr>
<td>• xxxhdpi</td>
<td>Extra-extra-extra high ~640 dpi</td>
</tr>
</tbody>
</table>

For more information on Android screen DPI settings, consult the official Android documentation page [https://developer.android.com/guide/practices/screens_support.html](https://developer.android.com/guide/practices/screens_support.html)

Example:

```json
"icons" : {
  "default" : "AndroidLauncher/icon-xhdpi.png",
  "mdpi" : "AndroidLauncher/icon-mdpi.png",
  "hdpi" : "AndroidLauncher/icon-hdpi.png",
  "xhdpi" : "AndroidLauncher/icon-xhdpi.png",
  "xxhdpi" : "AndroidLauncher/icon-xxhdpi.png",
  "xxxhdpi" : "AndroidLauncher/icon-xxxhdpi.png"
}
```

Default: Empty
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splash_screen</td>
<td>A map of splash screen override path(s) for each orientation and screen DPI type. All entries require a path that is either relative to &lt;engine&gt;\Code &lt;project&gt;\Resources or an absolute resource path to the PNG image.</td>
</tr>
<tr>
<td></td>
<td>Available sub-options (orientation type):</td>
</tr>
<tr>
<td></td>
<td>• land</td>
</tr>
<tr>
<td></td>
<td>• Map of overrides for the landscape orientation</td>
</tr>
<tr>
<td></td>
<td>• port</td>
</tr>
<tr>
<td></td>
<td>• Map of overrides for the portrait orientation</td>
</tr>
<tr>
<td></td>
<td>Available sub-options for each orientation type:</td>
</tr>
<tr>
<td></td>
<td>• default</td>
</tr>
<tr>
<td></td>
<td>• Default image used if a specific DPI is not specified</td>
</tr>
<tr>
<td></td>
<td>• mdpi</td>
</tr>
<tr>
<td></td>
<td>• Medium ~160 dpi</td>
</tr>
<tr>
<td></td>
<td>• hdpi</td>
</tr>
<tr>
<td></td>
<td>• High ~240 dpi</td>
</tr>
<tr>
<td></td>
<td>• xhdpi</td>
</tr>
<tr>
<td></td>
<td>• Extra high ~320 dpi</td>
</tr>
<tr>
<td></td>
<td>• xxhdpi</td>
</tr>
<tr>
<td></td>
<td>• Extra-extra high ~480 dpi</td>
</tr>
<tr>
<td></td>
<td>For more information on Android screen DPI settings, consult the official Android documentation page <a href="https://developer.android.com/guide/practices/screens_support.html">https://developer.android.com/guide/practices/screens_support.html</a></td>
</tr>
<tr>
<td>Example:</td>
<td>&quot;splash_screen&quot; :</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;land&quot; :</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;default&quot; :</td>
</tr>
</tbody>
</table>
|            |           "AndroidLauncher/splash-xhdpi.png",
|            |           "mdpi" :                                                                                                                        |
|            |           "AndroidLauncher/icon-mdpi.png",
|            |           "hdpi" :                                                                                                                        |
|            |           "AndroidLauncher/icon-hdpi.png",
|            |           "xhdpi" :                                                                                                                       |
|            |           "AndroidLauncher/icon-xhdpi.png",
|            |           "xxhdpi" :                                                                                                                      |
|            |           "AndroidLauncher/icon-xxhdpi.png"
|            |       },                                                                                                                                |
|            |       "port" :                                                                                                                           |

Version 1.28
<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{                                                                                           <strong>default</strong>: &quot;AndroidLauncher/icon-xhdpi.png&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;mdpi&quot; : &quot;AndroidLauncher/icon-mdpi.png&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;hdpi&quot; : &quot;AndroidLauncher/icon-hdpi.png&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;xhdpi&quot; : &quot;AndroidLauncher/icon-xhdpi.png&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;xxhdpi&quot; : &quot;AndroidLauncher/icon-xxhdpi.png&quot;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>Default: Empty</td>
</tr>
<tr>
<td><strong>app_public_key</strong></td>
<td>The application license key provided by Google Play. Required for using APK Expansion files or other Google Play Services.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>&quot;MIIBIjANBgkqhkiG9w0BAoUAgEAjvkl+K7tVASNkLExAmPlEOw6sCXV1Vx7</td>
</tr>
<tr>
<td></td>
<td>uV3IH5CQIZBRGT8KeYr6ThW1PhSMKMImj7KxjdYc1l8J0rwrVL3cm02ntnBEem5vRV6KxK</td>
</tr>
<tr>
<td></td>
<td>kDaFc5Fw6tJv3Sj6UvJYehB7tJupadfe9SVh0x3Zu2YsZLmbEJ</td>
</tr>
<tr>
<td></td>
<td>Hj7jG7YFYVqgou1W71QnExAmPlExi6mlsUJBFN4xALkNWZDI70i+D5</td>
</tr>
<tr>
<td></td>
<td>oS3KeVYKkd5WOU6IB8NnTY5VVdUOD4VPXrYmNY7FJZJMPuJLNVlArJ5SH/G0wUTR4ISI61AgJ</td>
</tr>
<tr>
<td></td>
<td>iQ1DAQAB&quot;</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;NoKey&quot;</td>
</tr>
<tr>
<td><strong>app_obfuscator_salt</strong></td>
<td>Application specific salt value for (un)obfuscation when using APK Expansion files</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>&quot;8d87473f5b24852836d6512abbd9e9b9869c208&quot;</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;&quot;</td>
</tr>
<tr>
<td><strong>enable_obb_in_dev</strong></td>
<td>Forces APK Expansion file mode in non-release builds. Value must be either &quot;true&quot; or &quot;false&quot;.</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;false&quot;</td>
</tr>
</tbody>
</table>
Using the Waf Build System

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_main_obb</td>
<td>Specify if the &quot;Main&quot; APK Expansion file should be used. This option toggles APK Expansion file mode in release builds. Value must be either &quot;true&quot; or &quot;false&quot;.</td>
</tr>
<tr>
<td>Default: &quot;false&quot;</td>
<td></td>
</tr>
<tr>
<td>use_patch_obb</td>
<td>Specify if the &quot;Patch&quot; APK Expansion file should be used. This option toggles APK Expansion file mode in release builds. Value must be either &quot;true&quot; or &quot;false&quot;.</td>
</tr>
<tr>
<td>Default: &quot;false&quot;</td>
<td></td>
</tr>
<tr>
<td>enable_key_screen_on</td>
<td>Enabled or disable the screen wake lock (device won't go to sleep while the application is running). Value must be either &quot;true&quot; or &quot;false&quot;.</td>
</tr>
<tr>
<td>Default: &quot;false&quot;</td>
<td></td>
</tr>
<tr>
<td>rc_pak_job</td>
<td>Path to the RC job XML file used to override the normal PAK files generation used in release builds. Path must be relative to &lt;lumberyard_version&gt;\Bin64\rc</td>
</tr>
<tr>
<td>Default: &quot;RcJob_Generic_MakePaks.xml&quot;</td>
<td></td>
</tr>
<tr>
<td>rc_obb_job</td>
<td>Path to the RC job XML file used to override the normal APK Expansion file(s) generation used in release builds. Path must be relative to &lt;lumberyard_version&gt;\Bin64\rc</td>
</tr>
<tr>
<td>Default: &quot;RCJob_Generic_Android_MakeObb.xml&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Waf Spec Files (*.json)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You use Waf spec files to specify which modules to include in a build configuration. All settings are mandatory if not explicitly stated otherwise.

A typical spec includes all modules that are required to build a game project. Lumberyard includes the following with the engine SDK:

- `game_and_engine.json` – Specs to build the sample game and engine
- `resource_compiler.json` – Specs to build the Resource Compiler
- `pipeline.json` – Specs to build the pipeline tools
- `all.json` – Specs to build all projects

The following is an example *.json file that illustrates a spec file layout:

```json
{
```

Version 1.28
74
```json
"description" : "Configuration to build the my game",
"visual_studio_name" : "My Game",
"comment"               : "This is meant to only compile tools on Windows.",
"disable_game_projects" : true,
"platforms" : ["win"],
"configurations" : ["debug","profile"],

"modules" :
[  
  "AzCore",
  "AzFramework",
  "AzToolsFramework",
  "GridMate",
  "LuaIDE",
  "Profiler"
]
}
```

**Note**

The `disable_game_projects` keyword does not compile the games specified in the `project.json` file. The default value is false, which means the specs compile the game projects by default.

**Platform-specific Entry Values**

You can apply the entry values in the table to targeted platforms and/or configurations. For example, a spec can build specific modules for win_x64 or a spec can build different modules in certain configurations.

- **modules** – Includes in the build all modules defined by this key, regardless of platform and configuration.
- **win_x64_modules** – Includes in the win_64 build all modules defined by this key, regardless of configuration.

Overlapping lists are combined into a single list based on the build command.

**Spec File Format Specification**

The general format of the JSON-based spec file is a dictionary of keyword values. The following table lists the possible keywords and their description.

<table>
<thead>
<tr>
<th>Keyword Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>Additional comments to add to the spec file.</td>
</tr>
<tr>
<td>configurations</td>
<td>The list of configurations that this spec supports. In other words, the spec only builds the modules listed in the spec if the current configuration exists in the list of configurations. This is an AND condition with the platforms value.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the spec file.</td>
</tr>
<tr>
<td>disable_game_projects</td>
<td>Flag that indicates that no game projects (as defined in project.json) are included in the build for this spec.</td>
</tr>
<tr>
<td>platforms</td>
<td>The list of platforms that this spec supports. In other words, the spec only builds the modules</td>
</tr>
</tbody>
</table>

Version 1.28
75
Using the Waf Build System

<table>
<thead>
<tr>
<th>Keyword Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>listed in the spec if the current target platform exists in this list of platforms.</td>
</tr>
<tr>
<td><strong>platform_configuration_defines</strong></td>
<td>•  <em>platform</em> and <em>configuration</em> are optional values.</td>
</tr>
<tr>
<td></td>
<td>•  Possible values for <em>platform</em> and <em>configuration</em> can be determined from the <em>waf_branch Spec</em> file.</td>
</tr>
<tr>
<td></td>
<td>•  The build uses the entry that matches the combination of these values and the build command.</td>
</tr>
<tr>
<td><strong>visual_studio_name</strong></td>
<td>Name of the generated Visual Studio solution that is used to distinguish this build spec from a build configuration.</td>
</tr>
</tbody>
</table>

**Waf Module Files (wscript)**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Wscript files are Python source files that have a fixed name and defined rules for the project folder. Waf picks up and processes the wscript file in each folder. Files can recurse into one or more subdirectories, define the build script for one or more modules, or both.

Wscript files are the main project script files for projects and can include the following:

- Specialized behavior for various Waf commands
- Different module types and entries
- Build rules for the folder
- Project- or target platform-specific definitions for compile, link, or other settings

Lumberyard includes a wscript file at the root folder that is used for the following:

- Loading all supported modules and tools relevant to a platform
- Importing all scripts necessary for configuring and building the engine
- Setting the available options that can be passed through the command line or in the default user options file located at _WAF_/user_settings.options
- Recursing into the Code and Engine folders at the root level

At the root is a compiled python script called *lmbr_waf.bat* that executes the Waf commands through the root wscript file.

**Lumberyard Engine Build Modules**

The Lumberyard Waf system includes the following predefined build modules that can help define the build rules for system modules:
<table>
<thead>
<tr>
<th>Build Module</th>
<th>Description</th>
<th>Consumers</th>
<th>Project Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CryConsoleApplication</td>
<td>Build module for generic console applications</td>
<td>ShaderCacheGen</td>
<td>Executable</td>
</tr>
<tr>
<td>CryDedicatedServer</td>
<td>Build module for dedicated (server) game project launchers</td>
<td>FeaturesTestsDedicatedLauncher</td>
<td>Executable</td>
</tr>
<tr>
<td>CryEditor</td>
<td>Build module for Lumberyard Editor project</td>
<td>Editor</td>
<td>Executable</td>
</tr>
<tr>
<td>CryEngineNonRCModule</td>
<td>Version of the CryEngineModule that does not attempt to create an RC file</td>
<td>CrySoundUnitTests, LyShine, AssetTaggingTools</td>
<td>Shared Library</td>
</tr>
<tr>
<td>CryEngineStaticModule</td>
<td>Build module to create static libraries</td>
<td>lua, md5, LZSS, Lzma, expat, DBAPI, zlib, lz4, PRT</td>
<td>Static Library</td>
</tr>
<tr>
<td>CryFileContainer</td>
<td>Build module that acts as a placeholder for source files</td>
<td>CryCommon, CryAudioCommon, EditorAudioControlsBrowser</td>
<td>Non</td>
</tr>
<tr>
<td>CryLauncher</td>
<td>Build module for game project launchers</td>
<td>FeaturesTestsLauncher</td>
<td>Executable</td>
</tr>
<tr>
<td>CryPipelineModule</td>
<td>Build module for pipeline components</td>
<td>CryExport2014, CryExport2015, CryExport2016, MayaCryExport22014, MayaCryExport22015, MayaCryExport22016</td>
<td>Custom</td>
</tr>
</tbody>
</table>
**Build Module** | **Description** | **Consumers** | **Project Type**  
---|---|---|---
CryPlugin | Build module for Lumberyard Editor plugins | AssetTagging, CryDesigner, EditorDesc, EditorAnimation, EditorFbxImport, EditorGameDatabase, SchematycPlugin | Shared Library  
CryPluginModule | Build module for Lumberyard Editor plugin modules | EditorCommon, PerforcePlugin | Shared Library  
CryResourceCompiler | Build module for the resource compiler application | ResourceCompiler | Executable  
CryStandAlonePlugin | Build module for Lumberyard Editor standalone plugins (does not link to any engine shared libraries) | EditorAudioControlsBrowser, EditorNoSound, EditorWwise, FBXPlugin, FFMPEGPlugin, MetricsPlugin, PrototypeEditorPlugin, StateMachineEditorPlugin, UiEditor | Shared Library  
CryUnitTestLauncher | Build module for unit test launchers | UnitTestLauncher | Executable

**Waf User Options and Settings**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In Lumberyard, the Waf options for configure and build time operations in are determined primarily by the following files in the *lumberyard_version*\dev\_WAF\_subdirectory:

- **default_settings.json** – Declares most of the configurable settings that control how Lumberyard and its game projects are built, including the default values.
- **Platform-specific settings** – Settings and defaults declared in the *settings* sections of *.json* files in the *lumberyard_version*\dev\_WAF\_settings\platforms subdirectory, including the following:
  - **common.android.json** – Options for Android.
  - **platform.darwin_x64.json** – Options for macOS.
  - **platform.ios.json** – Options for iOS.
Using the Waf Build System

- `user_settings.options` – Use this file to override default values defined in the `default_settings.json` file and platform-specific settings files.

**Topics**
- `default_settings.json` (p. 79)
- `user_settings.options` (p. 80)
- Waf User Settings (`user_settings.options`) (p. 80)

**default_settings.json**

The `default_settings.json` file and the `settings` sections of platform-specific `.json` files use the following format to organize related settings into groups:

```json
"Group Name": [
  {
    "long_form": "Command line form",
    "short_form": "Optional short form of the command line option",
    "attribute": "Waf option attribute",
    "default_value": "Default value",
    "description": "Brief description of the setting"
  },
  ...
],
```

Each option can have the following attributes:

- **long_form** – The long form of the command line argument that Waf accepts as a command-line override to the setting. This overrides both the default value and any override value in the `user_settings.options` file. The long form is usually preceded with a double hyphen (for example, `--enable-my-option`).
- **short_form** – (Optional) The short form of the command argument override option (for example, `-s`).
- **attribute** – The name of the attribute for configure and builds. The attribute is set to the Options WAF module object. This form of the attribute name is used in the `user_settings.options` override file.
- **default_value** – The default value if the setting is not overridden in the `user_settings.options` file or by the related command line argument.
- **description** – A brief description of the option.

The following example shows the `enabled_game_projects` attribute in the Game Projects section of the `default_settings.json` file.

```json
"Game Projects": [
  {
    "long_form": "--enabled-game-projects",
    "attribute": "enabled_game_projects",
    "default_value": "StarterGame,CloudGemSamples",
    "description": "Comma-separated list of game projects to enable for compiling"
  },
  ...
],
```

Version 1.28
user_settings.options

Use the user_settings.options file to override the default values specified in default_settings.json and platform-specific settings .json files. This configuration file is in a standard .cfg file format (not .json) with section names in brackets, as in the following example:

```
[Game Projects]
;enabled_game_projects = StarterGame,CloudGemSamples
```

The groupings are defined in the default_settings.json file and in the platform-specific settings .json files. The user_settings.options file uses the form of the attribute for each setting as it is defined in the corresponding .json file.

Overriding Default Values

The default values are commented out with a semi-colon ';' comment token at the beginning of each line in the user_settings.options file. To override a value, remove the semicolon from the beginning of the line and set the attribute to the value that you want. For example, the option to use IncrediBuild is off (False) by default. If you want to set it to True, modify the corresponding section of the user_settings.options file to resemble the following:

```
[IncrediBuild Options]
useIncrediBuild = True
;IncrediBuild_max_cores = 128
;auto_updateIncrediBuild_settings = False
;IncrediBuild_profile = Tools/build/waf-1.7.13/profile.xml
```

Overriding user_settings.options Changes

If you use the the user_settings.options file to override a default value like useIncrediBuild, you can override the change temporarily by using the command line argument defined for the setting, as shown in the following example:

```
lmbr_waf build_win_x64_vs2017_profile -p all --use-incredibuild=False
```

**Note**

Using the lmbr_waf command to override a value does not update the value in the user_settings.options file.

Waf User Settings (user_settings.options)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Global Waf build system settings are specified in the user_settings.options file located in the lumberyard_version\dev\WAF_subdirectory. If the user_settings.options file does not exist, Lumberyard uses the default_settings.json and platform-specific .json files to create a new one automatically. Every build that runs refers to this file to get the option values specific to the build.

The settings listed can be modified in the file directly, or through the Lumberyard WAF settings dialog box. To invoke the settings dialog box, enter show_option_dialog command into Waf as follows:

```
lmbr_waf show_option_dialog
```
The tables in the following sections describe the options available for override in the user_settings.options file. To override any setting, you can use the Override Parameter for the attribute. For more information, see Overriding user_settings.options Changes (p. 80).

Topics
- default_settings.json Options (p. 81)
- Output Folder Options (p. 86)
- Platform-Specific Options (p. 87)

### default_settings.json Options

The following tables describe the options defined in the default_settings.json file.

#### Build Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy_3rd_party_pdbs</td>
<td>--copy-3rd-party-pdb</td>
<td>Copies .pdb files from third party libraries for debugging.</td>
<td>True</td>
</tr>
<tr>
<td>crash_handler_token</td>
<td>--crash-handler-token</td>
<td>Token that you use to submit crash reports.</td>
<td></td>
</tr>
<tr>
<td>crash_handler_url</td>
<td>--crash-handler-url</td>
<td>Endpoint where you submit crash reports.</td>
<td></td>
</tr>
</tbody>
</table>
## Attribute

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_dynamic_file_globbing</td>
<td>--enable-dynamic-file-globbing</td>
<td>Enables globbing of files during all build operations, globbing only once during configure.</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> Excessive usage of file globbing negatively impacts build performance.</td>
<td></td>
</tr>
<tr>
<td>enable_link_time_optimization</td>
<td>--enable-link-time-optimization</td>
<td>If true, link time optimizations and code generation are enabled in the performance and release configurations.</td>
<td>False</td>
</tr>
<tr>
<td>enable_memory_tracking</td>
<td>--enable-memory-tracking</td>
<td>Enable the AZCORE_ENABLE_MEMORY_TRACKING define, which allows the Memory Driller to run and track all allocations.</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> This option severely impacts code execution times.</td>
<td></td>
</tr>
<tr>
<td>enable_msvc_timings</td>
<td>--enable-msvc-timings</td>
<td>Enable output timing information for msvc compile and link operations.</td>
<td>False</td>
</tr>
<tr>
<td>enable_whole_program_optimization</td>
<td>--enable-whole-program-optimization</td>
<td>If set to True, whole program optimizations are enabled for performance and release builds.</td>
<td>False</td>
</tr>
<tr>
<td>external_crash_reporting</td>
<td>--external-crash-reporting</td>
<td>Zip and upload symbols and build client with external crash reporting enabled. The value set is an additional build tag that is passed to the crash reporter system.</td>
<td></td>
</tr>
<tr>
<td>gems_optional</td>
<td>--gems-optional</td>
<td>Allows building of projects without gems.json files.</td>
<td>False</td>
</tr>
<tr>
<td>generate_debug_info</td>
<td>--generate-debug-info</td>
<td>Option to generate debug symbols and .pdb files for the build.</td>
<td>True</td>
</tr>
<tr>
<td>generate_map_file</td>
<td>--generate-map-file</td>
<td>Generate a map file during linking if the platform supports it.</td>
<td>False</td>
</tr>
<tr>
<td>generate_sig_debug_outputdelta</td>
<td>--sig-delta -s</td>
<td>Generate debug output showing signature differences that caused a task to rerun.</td>
<td>False</td>
</tr>
<tr>
<td>layout_binaries_only</td>
<td>--layout-binaries-only</td>
<td>Update only the binaries in a current layout. This supports programmer rapid iteration mode.</td>
<td>False</td>
</tr>
<tr>
<td>layout_hard_linking</td>
<td>--layout-hard-linking</td>
<td>If true, layouts are hard links, not full copies of files.</td>
<td>True</td>
</tr>
<tr>
<td>layout_include_pdbbs</td>
<td>--layout-include-pdbbs</td>
<td>When adding the binaries to the layout, include the .pdb files.</td>
<td>False</td>
</tr>
</tbody>
</table>
### Using the Waf Build System

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_parallel_link</td>
<td>--max-parallel-link</td>
<td>Controls the number of C++ linking operations that happen in parallel.</td>
<td>2</td>
</tr>
<tr>
<td>packaged_build_time</td>
<td>--packaged-build-time</td>
<td>A float value indicating the time that the build was packaged.</td>
<td></td>
</tr>
<tr>
<td>product_sku</td>
<td>-product-sku</td>
<td>Enables a project wide define PRODUCT_SKU_value. The specified value alters where .pak files are built to and read from and determines which platform resources are used in the gem\resources\ directory. For example, for a demo version of the game, you could set product_sku = demo. After PRODUCT_SKU_demo is defined, you can disable systems based on its value.</td>
<td>default</td>
</tr>
<tr>
<td>symbol_token</td>
<td>--symbol-token</td>
<td>Specify the token used for uploading symbols.</td>
<td></td>
</tr>
<tr>
<td>uber_file_size</td>
<td>--uber-file-size</td>
<td>Maximum content size of auto-generated uber files.</td>
<td>307200</td>
</tr>
<tr>
<td>upload_symbol_list</td>
<td>--upload-symbol-list</td>
<td>Specify the list of symbol patterns to upload for crash reporting.</td>
<td></td>
</tr>
<tr>
<td>use_crcfix</td>
<td>--use-crcfix</td>
<td>Use the crcfix tool to precompute CRCs in AZ_CRC macros.</td>
<td>True</td>
</tr>
<tr>
<td>use_debug_code_generator</td>
<td>--use-debug-code_generator</td>
<td>Uses the version of the code generator located in the Bin64xxxx.Debug directory instead of the Bin64 directory.</td>
<td>False</td>
</tr>
<tr>
<td>use_precompiled_header</td>
<td>--use-precompiled-header</td>
<td>Use a precompiled header for compilation where applicable.</td>
<td>True</td>
</tr>
<tr>
<td>use_uber_files</td>
<td>--use-uber-files</td>
<td>Use uber files for compilation.</td>
<td>False</td>
</tr>
<tr>
<td>version</td>
<td>--force-version</td>
<td>The version of the game project to embed in the game launchers.</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>win_build_renderer</td>
<td>--win-build-renderer</td>
<td>Specifies the type of renderer for a monolithic build. Possible values are DX11 or DX12.</td>
<td>DX11</td>
</tr>
</tbody>
</table>

### Deployment Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>deploy_projects_automatically</td>
<td>deploy-projects-automatically</td>
<td>Automatically runs the deploy command after each build.</td>
<td>True</td>
</tr>
</tbody>
</table>
# Game Projects

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled_game_projects</td>
<td>--enabled-game-projects</td>
<td>Comma-separated list of game projects to enable for compiling. By default, the possible values are:</td>
<td>CloudGemSamples, StarterGame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CloudGemDefectReportSample</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CloudGemSamples</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MultiplayerSample</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SamplesProject</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StarterGame</td>
<td></td>
</tr>
</tbody>
</table>

# IncrediBuild Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_update_incredibuild_settings</td>
<td>--auto-update-incredibuild-settings</td>
<td>Automatically attempts to update the registry for IncrediBuild, if needed. These registry updates are required to configure IncrediBuild to work properly with the Waf build system.</td>
<td>False</td>
</tr>
<tr>
<td>incredibuild_max_cores</td>
<td>--incredibuild-max-cores</td>
<td>Control the number of processes spawned by IncrediBuild.</td>
<td>128</td>
</tr>
<tr>
<td>incredibuild_profile</td>
<td>--incredibuild-profile</td>
<td>The IncrediBuild configuration value to load for IncrediBuild builds. If left blank, this value is generated automatically.</td>
<td></td>
</tr>
<tr>
<td>use_incredibuild</td>
<td>-i</td>
<td>Use IncrediBuild if available. Windows PC builds require at a minimum the Make and Build tools package. Android builds additionally require the Dev Tools Acceleration package.</td>
<td>False</td>
</tr>
</tbody>
</table>

# Misc Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap_third_party</td>
<td>--3rdpartypath</td>
<td>Optional parameter to pass the location of the \3rdParty directory as part of the bootstrap process.</td>
<td></td>
</tr>
<tr>
<td>bootstrap_tool_param</td>
<td>bootstrap-tool-param</td>
<td>Value set by Lumberyard Setup Assistant to inform Waf which platforms should be enabled.</td>
<td></td>
</tr>
</tbody>
</table>
### Attribute Override Parameter Description Default

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> This setting is automatically configured through Lumberyard Setup Assistant and should not be updated manually.</td>
<td></td>
</tr>
<tr>
<td>has_server_configs</td>
<td>--has-server-configs</td>
<td>Optional parameter to enable the _dedicated_server configurations.</td>
<td>True</td>
</tr>
<tr>
<td>has_test_configs</td>
<td>--has-test-configs</td>
<td>Optional parameter to enable the _test configurations.</td>
<td>True</td>
</tr>
<tr>
<td>max_cores</td>
<td>--max-cores</td>
<td>Number of parallel processes for local builds. To limit the number of cores used, set a specific value. A value less than or equal to 0 indicates that as many cores as needed will be used based on available hardware.</td>
<td>0</td>
</tr>
</tbody>
</table>

### Packaging Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy_assets</td>
<td>--always-copy-assets</td>
<td>An option specific to iOS and macOS. When running the packaging command, always copy any assets that are specified to the package, even for debug or profile builds.</td>
<td>False</td>
</tr>
<tr>
<td>package_projects_automatically</td>
<td>--package-projects-automatically</td>
<td>Automatically run the packaging command after each build, where available. This option is supported on Android, iOS, and macOS.</td>
<td>True</td>
</tr>
<tr>
<td>run_xcode_for_packaging</td>
<td>--runxcode-for-packaging</td>
<td>An option specific to iOS and macOS. When running the packaging command, run xcode_* from the command line to generate the app bundle resources for iOS and macOS platforms.</td>
<td>True</td>
</tr>
</tbody>
</table>

### Visual Studio Project Generator

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_project</td>
<td>--visual-studio-solution-default-project</td>
<td>The Visual Studio default project if not set in the solution user options (.suo) file.</td>
<td>Editor</td>
</tr>
<tr>
<td>msvs_version</td>
<td>--msvs-version</td>
<td>Version of the Visual Studio solution to generate when creating a new project with Project Configurator.</td>
<td>15</td>
</tr>
</tbody>
</table>
## Using the Waf Build System

### Attribute Override Parameter

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>specs_to_include_in_project_generation</td>
<td>--specs-to-include-in-project-generation</td>
<td>List of Waf spec files (p. 74) to include in the Visual Studio solution generation.</td>
<td>all, game, game_and_engine</td>
</tr>
<tr>
<td>visual_studio_solution_folder</td>
<td>--visual-studio-solution-folder</td>
<td>Name of the directory in which the generated Visual Studio solution should be stored.</td>
<td>Solutions</td>
</tr>
</tbody>
</table>

### Output Folder Options

Output folder options determine build output paths and folder name extensions for specific types of builds and environments.

The output path attributes in the Output Folder table can have configuration-based extensions from the Output Folder Name Extensions table. The output path attributes are autogenerated by Waf from the enabled platforms. The default values are defined by the default_folder_name attribute in each target's `lumberyard_version\dev\_WAF_\settings\platforms\platform.target.json` file.

#### Output Folder

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>out_folder_android_armv8_clang</td>
<td>--output-folder-android-armv8-clang</td>
<td>The base output folder name for the android_armv8_clang platform.</td>
<td>BinAndroidArmv8Clang</td>
</tr>
<tr>
<td>out_folder_ios</td>
<td>--output-folder-ios</td>
<td>Absolute or relative iOS target platform build output path.</td>
<td>BinIos</td>
</tr>
<tr>
<td>output_folder_darwin_x64</td>
<td>--output-darwin-x64</td>
<td>Absolute or relative macOS (Darwin) target platform build output path.</td>
<td>BinMac64</td>
</tr>
<tr>
<td>out_folder_win_x64_vs2017</td>
<td>--output-folder-win64-vs2017</td>
<td>The base output folder name for the win_x64_vs2017 platform.</td>
<td>Bin64vc141</td>
</tr>
<tr>
<td>out_folder_win_x64_vs2019</td>
<td>--output-folder-win64-vs2019</td>
<td>The base output folder name for the win_x64_vs2019 platform.</td>
<td>Bin64vc142</td>
</tr>
</tbody>
</table>

The following name extensions are appended to the output folder based on the target platform builds. These configuration extension options are autogenerated by Waf. The default values are defined by the default_output_ext attribute for each configuration in the `lumberyard_version\dev\_WAF_\settings\build_configurations.json` file.

#### Output Folder Name Extensions

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_folder_ext_debug</td>
<td>--output-folder-ext-debug</td>
<td>The output folder name extension for debug builds.</td>
<td>Debug</td>
</tr>
<tr>
<td>output_folder_ext_performance</td>
<td>--output-folder-ext-performance</td>
<td>The output folder name extension for performance builds.</td>
<td>Performance</td>
</tr>
</tbody>
</table>
## Attribute Override Parameter

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_folder_ext_profile</td>
<td>The output folder name extension for profile builds.</td>
</tr>
<tr>
<td>output_folder_ext_release</td>
<td>The output folder name extension for release builds.</td>
</tr>
</tbody>
</table>

## Platform-Specific Options

Specific settings for Android, iOS, macOS, and Windows are defined in files located in the `lumberyard_version\dev\_WAF\_settings\platforms\` directory as noted.

## All Platforms

The following settings define whether or not a platform is enabled to build. The default value comes from the value of the `enabled` key found in each platform-specific file in the `lumberyard_version\dev\_WAF\_settings\platforms\` directory.

### Enable Platform

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable_android_armv8_clang</td>
<td>--enable-android-armv8-clang</td>
<td>Enable the android_armv8_clang platform to build.</td>
<td>True</td>
</tr>
<tr>
<td>enable_win_x64_vs2017</td>
<td>--enable-win64-vs2017</td>
<td>Enable the win_x64_vs2017 platform to build.</td>
<td>True</td>
</tr>
<tr>
<td>enable_win_x64_vs2019</td>
<td>--enable-win64-vs2019</td>
<td>Enable the win_x64_vs2019 platform to build.</td>
<td>True</td>
</tr>
</tbody>
</table>

## Android

This section has been relocated to another part of the Lumberyard documentation, and is now part of the Reference for Android (p. 3187). See Waf settings (p. 3188).

## iOS

The following settings for iOS are defined in the `lumberyard_version\dev\_WAF\_settings\platforms\platform.ios.json` file.

### iOS Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ios_paks</td>
<td>--ios-paks</td>
<td>Forces .pak files to be built in non-release builds.</td>
<td>False</td>
</tr>
</tbody>
</table>
iOS Project Generator

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_ios_projects_automatically</td>
<td>--generate-ios-projects-automatically</td>
<td>Automatically generate an Xcode project for iOS.</td>
<td>True</td>
</tr>
<tr>
<td>ios_project_name</td>
<td>--ios-project-name</td>
<td>Name of the generated iOS project.</td>
<td>LumberyardiOSSDK</td>
</tr>
<tr>
<td>ios_project_folder</td>
<td>--ios-project-folder</td>
<td>Name of the directory in which the generated iOS projects should be stored.</td>
<td>Solutions</td>
</tr>
</tbody>
</table>

macOS

The following settings for macOS are defined in the `lumberyard_version\dev\WAF\settings\platforms\platform.darwin_x64.json` file.

Mac Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_build_monolithic</td>
<td>--mac-build-monolithic</td>
<td>Boolean flag to generate a monolithic or a non-monolithic build for macOS.</td>
<td>False</td>
</tr>
<tr>
<td>darwin_paks</td>
<td>--darwin-paks</td>
<td>Forces .pak files to be built in non-release builds.</td>
<td>False</td>
</tr>
</tbody>
</table>

Mac Project Generator

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_mac_projects_automatically</td>
<td>--generate-mac-projects-automatically</td>
<td>Automatically generate Xcode projects for macOS.</td>
<td>True</td>
</tr>
<tr>
<td>mac_project_name</td>
<td>--mac-project-name</td>
<td>Name of the generated project.</td>
<td>LumberyardSDK</td>
</tr>
<tr>
<td>mac_project_folder</td>
<td>--mac-project-folder</td>
<td>Name of the directory in which the generated macOS projects should be stored.</td>
<td>Solutions</td>
</tr>
</tbody>
</table>

Windows

The following settings for Visual Studio are defined in the `lumberyard_version\dev\WAF\settings\platforms\common.win_msvc.json` file.
Visual Studio Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_vs_solution_automatically</td>
<td>--generate-vs-solution-automatically</td>
<td>Automatically generate Visual Studio solutions during every configure.</td>
<td>True</td>
</tr>
</tbody>
</table>

The following settings for Visual Studio 2017 are defined in the `lumberyard_version\dev\WAF\settings\platforms\platform.win_x64_vs2017.json` file.

Visual Studio 2017 Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_vs2017_projects_automatically</td>
<td>--generate-vs2017-projects-automatically</td>
<td>Automatically generate Visual Studio 2017 projects and solutions during every configure if the Windows VS2017 platform is enabled.</td>
<td>True</td>
</tr>
<tr>
<td>vs2017_solution_name</td>
<td>--vs2017-solution-name</td>
<td>The name of the generated Visual Studio 2017 solution.</td>
<td>LumberyardSDK_vs2017</td>
</tr>
<tr>
<td>win_vs2017_vcvarsall_args</td>
<td>--win-vs2017-vcvarsall-args</td>
<td>Additional arguments to pass to the vcvarsall.bat file.</td>
<td></td>
</tr>
<tr>
<td>win_vs2017_vswhere_args</td>
<td>--win-vs2017-vswhere-args</td>
<td>The arguments to pass to vswhere when locating Visual Studio 2017 executables. The default maximum is up to but not including the next major version of Visual Studio. The default minimum is the last known ABI incompatibility for Lumberyard builds.</td>
<td>-version [15.9.28307.770,16.0)</td>
</tr>
<tr>
<td>win_vs2017_winkit</td>
<td>--win-vs2017-winkit</td>
<td>The windows kit that Visual Studio 2017 builds Windows targets against.</td>
<td></td>
</tr>
</tbody>
</table>

The following settings for Visual Studio 2019 are defined in the `lumberyard_version\dev\WAF\settings\platforms\platform.win_x64_vs2019.json` file.

Visual Studio 2019 Options

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Override Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Override Parameter</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>win_vs2019_vswere_args</td>
<td>--win-vs2019-vswere-args</td>
<td>The arguments to pass to vswere when locating Visual Studio 2019 executables. The default maximum is up to but not including the next major version of Visual Studio. The default minimum is the last known ABI incompatibility for Lumberyard builds.</td>
<td>-version [16.2.29230.47,17.0]</td>
</tr>
</tbody>
</table>

**Waf Commands and Options**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Contents**

- Waf Configuration (p. 90)
- Build Configuration (p. 91)
- Multiplayer Configuration (p. 95)

Before building a project using Waf, you must run `configure` from the command line. The `configure` command recursively processes all of the wscript configuration files starting from the root directory and generates a Visual Studio solution file for the entire project. You can set an option to generate a solution file during the `configure` command.

**Note**

The Waf script automatically runs Lumberyard Setup Assistant to ensure the correct third-party libraries are available and the proper links are created to compile the game code, engine and asset pipeline, and editor and tools.

**Waf Configuration**

To run the Waf executable, run the following command at the `lumberyard_version\dev\` directory of your project:

`lmbr_waf configure`

This command iterates through all the Waf project configuration files and sets up the project-specific settings in the Waf cache, which is used in subsequent build commands. It also uses the host environment to determine which platforms are available to build.
Example

The following example shows the output of the `lmbr_waf configure` command:

```
[WAF] Executing 'configure'
Running SetupAssistant.exe...
---- Lumberyard Setup Assistant ----
SDK location: d:/lumberyard_engine/dev
Third party location: d:/lumberyard_engine/dev/3rdParty
Capabilities Available, [x] enabled - [ ] disabled:
[ ] rungame - Run your game project
[ ] runeditor - Run the Lumberyard Editor and tools
[X] compilegame - Compile the game code
[X] compileengine - Compile the engine and asset pipeline
[X] compileeditor - Compile the Lumberyard Editor and tools
[ ] compileandroid - Compile for Android devices
[ ] compileios - Compile for iOS devices
Successfully executed
[INFO] Configure "win_x64_vs2017 - [debug, profile, performance, release, debug_dedicated, performance_dedicated, release_dedicated]"
[INFO] Configure "win_x64_vs2019 - [debug, profile, performance, release, debug_dedicated, performance_dedicated, release_dedicated]"

[WAF] 'configure' finished successfully (10.335s)
[WAF] Executing 'generate_uber_files' in 'd:\ws\lyengine\dev\BinTemp'
[WAF] 'generate_uber_files' finished successfully (2.177s)
[WAF] Executing 'msvs' in 'd:\ws\lyengine\dev\BinTemp'
```

The `configure` command uses the settings defined in the `user_settings.options` file that is located in the `lumberyard_version\dev\_WAF_` subfolder. You can edit this file in a text editor or enter the following command to use the Lumberyard Waf settings tool:

```
lmbr_waf show_option_dialog
```

By default, whenever the `configure` command is run, Visual Studio solution and project files are created in the `dev\Solutions` directory. The name of the solution file includes the version of Visual Studio for which it has been generated, such as `LumberyardSDK_vs2017.sln`. You can change this behavior, the solution directory, and the solution name in the `user_settings.options` file.

You can also generate the solution files manually by running the command `lmbr_waf msvs`, or the version-specific `lmbr_waf msvs_2017`.

Build Configuration

After configuring Waf, you can run the `build` command.

The following example shows the syntax:

```
lmbr_waf build_platform_configuration -p spec
```

The following commands and options are available:

- `configure` – Must be run before any clean or build command. Loads all modules, configs, and project specs; validates and sets up the working cached build Python file.
- `build_*` – Builds the specified project spec for the specified platform and configuration.
- `package_*` – Creates a runnable package of the specified project spec for the specified platform and configuration for supported platforms. Supported operating systems and devices include Android, iOS, and macOS.
Using the Waf Build System

- **deploy_*** – Deploys the specified project spec for the specified platform and configuration to a remote device for supported platforms. Supported operating systems and devices include Android.
- **clean_*** – Cleans out intermediate and target files that were generated for the particular platform and configuration.

The following example shows how to build release for Windows x64 with Visual Studio 2017:

```
lmbr_waf build_win_x64_vs2017_release -p all
```

**Note**
Combining the `clean_*` and `build_*` commands is the equivalent of performing a rebuild.

**Configure command options**

<table>
<thead>
<tr>
<th>Command</th>
<th>Command Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_*</td>
<td><code>-p spec_name</code></td>
<td>The spec name to use to build or clean a project.</td>
</tr>
<tr>
<td>package_*</td>
<td><code>--project-spec=spec_name</code></td>
<td>Optional flag to filter on which targets to build. The targets must be included in the project spec above in order for this to work.</td>
</tr>
<tr>
<td>clean_*</td>
<td><code>--targets=target1,target2,...</code></td>
<td>Optional flag to filter on which targets to build. The targets must be included in the project spec above in order for this to work.</td>
</tr>
<tr>
<td>build_*</td>
<td>`--profile-execution=(True</td>
<td>False)`</td>
</tr>
<tr>
<td>clean_*</td>
<td><code>--execsolution=VS_solution_path</code></td>
<td>This internally-generated command line is a Visual Studio solution that provides a way to build Waf commands invoked from the VS IDE to apply additional overrides that can be defined in the .vcxproj files themselves.</td>
</tr>
<tr>
<td>config</td>
<td><code>--file-filter=source_files</code></td>
<td>An option to pass in a comma-separated list of absolute paths to source files to filter the build on. This option is useful to build specific files.</td>
</tr>
<tr>
<td>build_*</td>
<td>`--show-includes=(True</td>
<td>False)`</td>
</tr>
<tr>
<td>build_*</td>
<td>`--show-preprocessed-file=(True</td>
<td>False)`</td>
</tr>
<tr>
<td>build_*</td>
<td>`--show-disassembly=(True</td>
<td>False)`</td>
</tr>
</tbody>
</table>
Using the Waf Build System

<table>
<thead>
<tr>
<th>Command</th>
<th>Command Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configure</td>
<td>--update-settings=(True</td>
<td>False)</td>
</tr>
</tbody>
</table>

You can set the command options at build time. These options override the values set in the user_settings.options file. For more information, see Project Configurator (p. 80).

Only modules that support each project configuration are built from the project spec. If a module is defined in the spec that only can be built in debug or profile, building in performance mode excludes that project from compilation.

Project configurations parameters

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Asserts</th>
<th>Profiling</th>
<th>Optimization</th>
<th>Logging</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>Yes</td>
<td>All</td>
<td>Minimum</td>
<td>Yes</td>
<td>Slowest – Focuses on debugging with asserts enabled, all profiling features enabled, and logging enabled.</td>
</tr>
<tr>
<td>profile</td>
<td>No</td>
<td>All</td>
<td>Medium</td>
<td>Yes</td>
<td>Fast – Strikes a balance between debugging and performance with all profiling features and logging enabled.</td>
</tr>
<tr>
<td>performance</td>
<td>No</td>
<td>Few</td>
<td>Maximum</td>
<td>No</td>
<td>Very fast – Performance similar to release but has some profiling features enabled; difficult to debug; no logging.</td>
</tr>
<tr>
<td>release</td>
<td>No</td>
<td>None</td>
<td>Maximum</td>
<td>No</td>
<td>Fastest – Highest performance; most difficult to debug; no profiling features; no logging.</td>
</tr>
</tbody>
</table>

Build command project spec options

<table>
<thead>
<tr>
<th>Spec</th>
<th>Platform</th>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>win_x64_vs2017, win_x64_vs2019, darwin_x64</td>
<td>Debug, profile, performance, release</td>
<td>Configuration to build the engine, editor, plugins, and tools</td>
</tr>
<tr>
<td>game_and_engine</td>
<td>win_x64_vs2017, win_x64_vs2019, darwin_x64, linux_x64</td>
<td>Debug, profile, performance, release</td>
<td>Configuration to build the engine and game project</td>
</tr>
</tbody>
</table>
Using the Waf Build System

<table>
<thead>
<tr>
<th>Spec</th>
<th>Platform</th>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcc_plugins</td>
<td>win_x64_vs2017, win_x64_vs2019</td>
<td>Debug, profile</td>
<td>Configuration to build tools for the asset pipeline</td>
</tr>
<tr>
<td>resource_compiler</td>
<td>win_x64_vs2017, win_x64_vs2019</td>
<td>Debug, profile</td>
<td>Configuration to build the Resource Compiler only</td>
</tr>
</tbody>
</table>

**Build configuration options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--progress</td>
<td>Shows the build progress and updates in real time.</td>
</tr>
<tr>
<td>--project-spec</td>
<td>Specifies the project spec to use when cleaning or building the project.</td>
</tr>
<tr>
<td>--show-includes</td>
<td>Shows the includes for each compiled file.</td>
</tr>
<tr>
<td>--target</td>
<td>Specifies the target to build and its dependencies. The target must exist in the specified project spec; otherwise, all targets in the project spec are built.</td>
</tr>
</tbody>
</table>

**Command Chaining Options**

<table>
<thead>
<tr>
<th>Command</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_*</td>
<td>--package-projects-automatically=(True</td>
<td>False)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported platforms include Android, iOS, and macOS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following example runs only the build command:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>lmbr_waf -p all build_android_armv8_clang_profile --package-projects-automatically=True</code></td>
</tr>
<tr>
<td>build_*</td>
<td>--deploy-platform_root=(True</td>
<td>False)</td>
</tr>
<tr>
<td>package_*</td>
<td></td>
<td>Supported platforms include Android.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>platform_root</code> example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>android_armv8_clang =&gt; android (--deploy-android)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following example runs only the package command:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>lmbr_waf package_android_armv8_clang_profile --deploy-android=False</code></td>
</tr>
</tbody>
</table>
Using the Waf Build System

### Command Option Description

The following command ensures that all three commands (`build`, `package`, and `deploy`) run:

```
lmbr_waf -p all build_android_armv8_clang_profile --package-projects-automatically=True --deploy-android=True
```

### Multiplayer Configuration

Before you can build multiplayer information, you must build the dedicated server. This creates a directory called `Bin64.Dedicated` that includes the binaries directory and configuration files for the dedicated server.

To build the dedicated server, run the following build command for your version of Visual Studio:

```
lmbr_waf build_win_x64_vs2017_profile_dedicated -p dedicated_server
```

### Waf Supported Operating Systems and Compilers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This topic provides information about the operating systems and compilers that Waf supports. For more information about supported configurations, see Waf Commands and Options (p. 90).

#### Supported operating systems

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Build environment</th>
<th>Waf short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit Windows</td>
<td>MSBuild / Visual Studio 2017</td>
<td>win_x64_vs2017</td>
</tr>
<tr>
<td>64-bit Windows</td>
<td>MSBuild / Visual Studio 2019</td>
<td>win_x64_vs2019</td>
</tr>
</tbody>
</table>

The following compilers are supported based on the build operating system.

#### Supported compilers

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Windows 64-Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSVC 10.0 (Visual Studio 2010)</td>
<td>Yes (only for CryExport2014)</td>
</tr>
<tr>
<td>MSVC 11.0 (Visual Studio 2012)</td>
<td>Yes (only for CryExport2015)</td>
</tr>
<tr>
<td>MSVC 12.0 (Visual Studio 2013)</td>
<td>Yes (except for CryExport2014 and CryExport2015)</td>
</tr>
<tr>
<td>MSVC 14.0 (Visual Studio 2015)</td>
<td>Yes (except for CryExport2014 and CryExport2015)</td>
</tr>
</tbody>
</table>
Using the Waf Build System

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Windows 64-Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSVC 15.0 (Visual Studio 2017)</td>
<td>Yes (except for CryExport2014 and CryExport2015)</td>
</tr>
<tr>
<td>Clang</td>
<td>No</td>
</tr>
</tbody>
</table>

**Waf Project Settings**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

When defining a project's build settings (wscript), you can specify several different project settings for the build modules to configure the correct parameters for the project.

The following table provides the valid attributes for the different build modules.

**Build attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Target to Platform or Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>additional_manifests</td>
<td>Additional manifests to add to MSVC applications</td>
<td>Y</td>
</tr>
</tbody>
</table>
| additional_settings         | Container that groups compile settings and acts upon them recursively; useful for specifying options for particular files in a project For example, you can disable precompiled headers for a specific file using the following: 
```... additional_settings = Settings ( files = 'my_file.cpp', disable_pch=True ) ``` | Y                                   |
<p>| build_in_dedicated          | True by default; if False, the module will not be built when building in dedicated server mode | N                                   |
| cflags                      | Additional C flags to pass to the compiler                                 | Y                                   |
| cxxflags                    | Additional CXX flags to pass to the compiler                               | Y                                   |
| defines                     | List of additional pre-processor defines for the project                   | Y                                   |
| enable_rtti                 | Enable RTTI for a particular module                                        | Y                                   |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Target to Platform or Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>export_defines</td>
<td>List of definitions to add to the module that has a 'use' dependency on the current module</td>
<td>Y</td>
</tr>
<tr>
<td>export_definitions</td>
<td>List of export definitions to export using the /DEF: compiler option</td>
<td>Y</td>
</tr>
<tr>
<td>export_includes</td>
<td>List of additional include paths to add to the module that has a 'use' dependency on the current module</td>
<td>Y</td>
</tr>
<tr>
<td>features</td>
<td>Additional custom features to apply to the project during the build</td>
<td>Y</td>
</tr>
<tr>
<td>file_list</td>
<td>List of file specs that contain the files to include in the project</td>
<td>Y</td>
</tr>
<tr>
<td>files</td>
<td>List of files to include for the module</td>
<td>N</td>
</tr>
<tr>
<td>force_dynamic_crt</td>
<td>Forces the use of dynamic runtime CRT for the project</td>
<td>N</td>
</tr>
<tr>
<td>force_static_crt</td>
<td>Forces the use of static runtime CRT for the project</td>
<td>N</td>
</tr>
<tr>
<td>framework</td>
<td>(Darwin) Specifies the framework to use</td>
<td>Y</td>
</tr>
<tr>
<td>frameworkpath</td>
<td>(Darwin) Specifies additional paths to search for frameworks</td>
<td>Y</td>
</tr>
<tr>
<td>includes</td>
<td>Additional include paths for the module</td>
<td>Y</td>
</tr>
<tr>
<td>lib</td>
<td>Additional input libraries to link against</td>
<td>Y</td>
</tr>
<tr>
<td>libpath</td>
<td>Additional library paths for the module</td>
<td>Y</td>
</tr>
<tr>
<td>linkflags</td>
<td>Additional linker flags to pass to the linker</td>
<td>Y</td>
</tr>
<tr>
<td>meta_includes</td>
<td>Additional meta includes for WinRT using the /AI compiler option</td>
<td>Y</td>
</tr>
<tr>
<td>need_deploy</td>
<td>Hint to deploy the module before debugging in Visual Studio</td>
<td>N</td>
</tr>
<tr>
<td>output_sub_folder</td>
<td>Optional subfolder under the target output folder in which to copy the module binary</td>
<td>N</td>
</tr>
<tr>
<td>pch</td>
<td>Specifies the precompiled header (PCH) file, if in use</td>
<td>N</td>
</tr>
<tr>
<td>platforms</td>
<td>List of platforms to restrict the module to build on; if missing, a specific platform will not be targeted at the project definition level</td>
<td>N</td>
</tr>
<tr>
<td>priority_includes</td>
<td>Same as the includes paths, except this include list is added prior to the ones defined in the includes paths</td>
<td>N</td>
</tr>
</tbody>
</table>
### Using the Waf Build System

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Target to Platform or Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>List of source files to add directly to the project</td>
<td>N</td>
</tr>
<tr>
<td>target</td>
<td>Project name of the target</td>
<td>N</td>
</tr>
<tr>
<td>use</td>
<td>List of static library modules that are part of the Waf build to which you can add dependencies and static links</td>
<td>Y</td>
</tr>
<tr>
<td>uselib</td>
<td>Add a dependency on a 3rd party library by name</td>
<td>Y</td>
</tr>
<tr>
<td>vs_filter</td>
<td>Folder filter in the generated solution file where this project exists</td>
<td>N</td>
</tr>
</tbody>
</table>

### Platform and Configuration Targeting

If allowed (refer to the third column in the table above), you can set an attribute value to apply only under certain target platforms and configurations. Each attribute can be universal for all builds or targeted specifically to a platform/configuration combination:

- `[Attribute]` – Applies to any target platform/configuration for the attribute
- `[target_platform]_[attribute]` – Applies to any configuration for a specific target platform for the attribute
- `[configuration]_[attribute]` – Applies to a specific configuration for any target platform for the attribute
- `[target_platform]_[configuration]_[attribute]` – Applies to a specific target platform and configuration for the attribute

### Features

The Lumberyard Waf system allows the use of custom features to add functionality to a project's build pipeline.

#### Build features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt5</td>
<td>Passes files through the QT5 moc processor</td>
</tr>
<tr>
<td>generate_rc_file</td>
<td>Creates an RC file and copies the resources, such as the icon file; win_x64 only</td>
</tr>
<tr>
<td>wwise</td>
<td>Sets the following for building and linking against Wwise: environment, includes, libraries, and library paths</td>
</tr>
<tr>
<td>GoogleMock</td>
<td>Sets the following for building and linking against Google Mock: environment, includes, libraries, and library paths</td>
</tr>
<tr>
<td>AWSNativeSDK</td>
<td>Sets the following for building and linking against the AWS Native SDK library: environment, includes, libraries, and library paths</td>
</tr>
</tbody>
</table>
### Using the Waf Build System

#### AWSGameLift
Sets the following for building and linking against the AWS GameLift library: environment, includes, libraries, and library paths

#### GridMate
Sets the following for building and linking against the GridMate library: environment, includes, libraries, and library paths

### Using the Waf Artifacts Cache

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Waf artifacts cache is for Waf-generated files. The cache speeds up Waf build time by caching previously built artifacts and retrieving them when the task signature hasn't changed. To calculate the task signature, Lumberyard uses MD5 hashes of the task's run function, source file, dependencies, and build environment.

#### Improvements in Build Time

The Waf artifacts cache can significantly improve clean, non-incremental build time. The following table shows, in minutes, typical differences in build times for packaging jobs.

<table>
<thead>
<tr>
<th>Operating System or Device</th>
<th>Clean Build Time Without Cache</th>
<th>Clean Build Time with Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>75</td>
<td>14</td>
</tr>
<tr>
<td>macOS</td>
<td>163</td>
<td>25</td>
</tr>
</tbody>
</table>

#### Supported File Types

Unlike the normal compiler cache, which caches only `.obj` files, the Waf artifacts cache caches all files that Waf generates. It supports the following file extensions: `.h`, `.cpp`, `.json`, `.o`, `.obj`, `.inl`, `.inline`, `.rc`, `.res`, `.moc`, `.lib`, `.dll`, `.exp`, `.pdb`, `.manifest`, and `.exe`.

#### Limitations and Notes

Keep in mind the following limitations and considerations about the Waf artifacts cache.

- The Waf artifacts cache can be used among multiple machines that have different build paths. However, to correctly retrieve files from the cache, all builds must use the same version compiler and linker.
- The Waf artifacts cache doesn't support the `/Zi` compiler flag. Use the `/Z7` compiler flag instead.

The Waf artifacts cache is implemented at the Waf level, not at the compiler/linker level. When you use the `/Zi` flag, the compiler generates a separate `.pdb` file for the `.obj` file. However, because Waf does not know the `.pdb` file name in advance, it cannot upload or retrieve the `.pdb` files. When you use the `/Z7` flag instead, the symbolic debugging information is stored in the `.obj` file itself.
• The compiler assumes the same compilation environment when you use a `.pch` file. Therefore, caching precompiled headers across different workspace paths is not supported. For more information, see Precompiled Header Consistency Rules in the Microsoft Visual Studio documentation.

• When caching for different workspace paths is enabled, builds with and without Waf artifacts caching have different task signatures. Thus, if you run a build with Waf cache and then run another build without Waf cache, all files are recompiled.

## Waf Options

The Waf artifacts cache feature has the following Waf options:

---artifacts-cache -- Specifies a string that represents the artifacts cache path. The cache path can be any directory path on local disk or on a net share. The default value is the empty string ("""). If the option is not set or empty, the artifacts cache is not used.

---artifacts-cache-restore -- If true, the task's target output files are copied from the artifacts cache before the task is run. The default value is false.

---artifacts-cache-upload -- If true, the task's target output files are uploaded to the artifacts cache after the task is finished. The default value is false.

### Example

The following command specifies the artifacts cache path as `E:\waf_artifacts_cache` and restores artifacts from cache on cache hit. On cache miss, the command uploads the task's target output files to the artifacts cache after the task is finished.

```
lmbr_Waf build_win_x64_vs2017_fullprofile -p all --artifacts-cache="E:\waf_artifacts_cache" --artifacts-cache-restore=True --artifacts-cache-upload=True
```

To build using Visual Studio, update the `dev/_WAF_/user_settings.options` file to enable the artifacts cache as in the following example:

```
artifacts_cache = E:/artifacts_cache
artifacts_cache_restore = True
artifacts_cache_upload = True
```

## Deleting State Artifacts

Use the `clean_stale_cached_artifacts` Waf command to delete stale artifacts from the cache. The command has the following options:

---artifacts-cache-days-to-keep -- Number of days to keep the artifacts before they are considered stale.

---artifacts-cache-wipeout -- If true, deletes all artifacts from the cache.

### Examples

The following example specifies that artifacts in the cache location `E:\waf_artifacts_cache` be kept for four days.

```
lmbr_waf clean_stale_cached_artifacts --artifacts-cache="E:\waf_artifacts_cache" --artifacts-cache-days-to-keep=4
```

The following example deletes all artifacts from the cache location `E:\waf_artifacts_cache`. 

Version 1.28

100
lmbr_waf clean_stale_cached_artifacts --artifacts-cache="E:\waf_artifacts_cache" --artifacts-cache-wipeout=True

How the Waf Artifacts Cache Works

The Waf cache feature helps ensure that the proper task uid and env signatures are used.

Waf Pickle File

The Waf pickle file stores information about Waf targets.

For a clean build, the Waf cache feature uses a signature calculated before run() is called to decide the task's runnable status. However, because of changes in node dependencies, outputs or build environment, the task's signature can change between the calls to run() and post_run(). Merging the cached Waf pickle file with the local Waf pickle file helps to ensure that Waf can get the right dependencies for a task. It also helps to ensure that Waf can get the signature that matches the task signature from cache.

The Waf pickle file has the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>The node that represents the root of the file system.</td>
<td>Node.</td>
</tr>
<tr>
<td>node_deps</td>
<td>Implicit dependencies.</td>
<td>Dictionary mapping of a task-computed uid to nodes.</td>
</tr>
<tr>
<td>raw_deps</td>
<td>Implicit file dependencies that could not be resolved.</td>
<td>Dictionary mapping of a task-computed uid to any serializable type.</td>
</tr>
<tr>
<td>task_sigs</td>
<td>Signature of the tasks that are run.</td>
<td>Dictionary mapping of a task-computed uid to a hash that represents the task's signature.</td>
</tr>
<tr>
<td>azcg</td>
<td>Data for AzCodeGen tasks.</td>
<td>Dictionary mapping of a task-computed uid to a list of AzCodeGen saved data.</td>
</tr>
<tr>
<td>cached_engine</td>
<td>Engine path for the build that created the cached pickle file.</td>
<td>String</td>
</tr>
<tr>
<td>cached_tp_root</td>
<td>Third-party path for the build that created the cached pickle file.</td>
<td>String</td>
</tr>
</tbody>
</table>

Loading Data from a Waf Pickle File

To use the Waf artifacts cache, Waf must have the latest build pickle data from a cached pickle file. It then must merge that data with the local build pickle data. If an entry with the task uid key appears in both local data and cached data, then the local one is used. If the data merged from both pickle files contains node instances from the cached build which are unusable in current build, those node instances are recreated for the current build.

The following diagram shows this workflow.
Initiate BuildContext

Load build data from local pickle file if exists

Load build data from cached pickle file if exists, delete cache_sig from all nodes.

Merge two build data (like task signatures, task dependencies), if there is value conflict, then keep the value from local data.

Recursively find all the node instances that come from the cached build, and recreate the nodes for current build.

Task producer starts
Overriding the Task’s uid and env Signatures

When using the Waf artifacts cache feature across multiple machines with different paths, the task's uid and env signature shouldn't depend on an absolute path. When the Waf artifacts cache is enabled, the calculation of task's uid and env signatures is overridden and the absolute path is converted to a relative path.

Executing Tasks

When the Waf artifacts cache is enabled, it tries to retrieve the target files from the cache before it runs a task. If a cache miss or an error occurs during the file transfer, it runs the task and uploads the generated target files to the Waf artifacts cache after post_run() is called.

When the target files are successfully copied from the Waf artifacts cache, Waf skips the run function and calls post_run(). Because Lumberyard customizes the run and post_run functions for each task type, it can update a task's dependencies, outputs, or build environment. This changes the task's signature. When the run function is skipped, Lumberyard updates the task's dependencies, outputs, or build environment in the post_run function to keep the signature consistent.

For example, during the run function, the AzCodeGen task updates the task's INCPATHS and creates new linking tasks. If the AzCodeGen task's target output files are retrieved from cache successfully, the run function is skipped. Then Lumberyard uses the post_run function to update the task's INCPATHS and create new linking tasks.

The following diagram shows this workflow.
Adding Third-Party Libraries

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Gems (p. 1064) are Lumberyard extensions that can be enabled or disabled for any game project. This modularity is advantageous when you want to add a third-party library to specific games. You can add the third-party library to a new or existing gem, and then enable the gem for the projects that you want. This narrows the scope of the library to only the games that use the gem.

Adding a third-party library to a gem includes the following steps:

1. Determine the location of the library's content files, including its include headers, static libraries, and/or shared libraries.
2. Create a definition file for the library.
3. Configure other gems or modules to use the library.
4. Run `lmbr_waf configure` to process the library definition files.

In the following example, the steps add a fictional third-party library to a gem. The library is a static library called "SuperLibrary" that supports both Windows and macOS. The gem is called "MyGem".

**Step 1. Determine the Location of the Third-Party Library**

When you add a third-party library to a gem, you specify two locations: one for the library definition, and one for the library contents.

**Definition File**

The gem module expects the third-party SDK .json definition file (in this example, `superlibrary.json`) to reside in a subdirectory named `3rdParty`. The following gem directory structure shows the location of the `lumberyard_version\dev\Gems\gem_name\3rdParty` directory.

```
+---\3rdParty
+---\Assets
+---\Code
|   +---\wscript
+---\External
+---\gem.json
```

**Library Files**

The library code file location is not predetermined, but a best practice is to place them inside the gem's directory structure in an `External\SDK_name` directory, like this:

```
lumberyard_version\dev\Gems\gem_name\External\SDK_name
```

The following directory structure shows the `External` directory with the `SuperLibrary` directory inside.

```
+---\External
    +---\SuperLibrary
        +---\includes
            +---superlibrary.h
```
Step 2. Create a Third-Party Library Definition File

Create a .json definition file for the third-party library. Using the directory structure from the previous step, the third-party definition file superlibrary.json looks like this:

```json
{
    "name": "SUPERLIBRARY",
    "source": "@GEM@/External/SuperLibrary",
    "description": "My Super Library",
    "includes": [
        "Include"
    ],
    "defines": [],
    "lib_required": "True",
    "platform": {
        "win_x64_vs2017": {
            "libpath": [
                "lib/win32"
            ],
            "lib": [
                "superlibrary.lib"
            ]
        },
        "win_x64_vs2019": "@win_x64_vs2017",
        "darwin_x64": {
            "libpath": [
                "lib/osx"
            ],
            "lib": [
                "superlibrary.a"
            ]
        }
    }
}
```

Note

- The `source` key contains an alias called `@GEM@`. The `@GEM@` alias represents the root path of the current gem. The `source` key specifies the base of the third-party library directory on which the other paths specified by keys like `includes` and `libpath` are based.
- The `platform` key sections describe the locations of platform-specific library files based on the library identifier.

For more information, see Creating Third-Party Library Configuration Files for Waf (p. 108).

Step 3. Apply the Library to Modules or Gems That Require It

By default, after the third-party library definition is set in the gem, it is automatically available to that gem. In order for this gem to be available in other modules, those modules need to add `SUPERLIBRARY` to their `uselib` list.

Other gems (or the game gem) that require access to the gem-scoped third-party library must establish a dependency on the gem that has the library. For example, if you want a gem called `OurGem` to have...
access to the SuperLibrary library in MyGem, OurGem's gem.json file must define a dependency on MyGem.

**Example**

In the OurGem.gem.json file, OurGem declares a dependency on MyGem:

```json
{
    "Dependencies": [
        {
            "Uuid": "981435f1646a4ccfbd7733920c011b6",
            "VersionConstraints": [
                ">=0.1"
            ],
            "_comment": "MyGem"
        }
    ],
    "GemFormatVersion": 3,
    "Uuid": "1daafa0ea544f64befb74a1cc719a9c",
    "Name": "OurGem",
    "DisplayName": "Our Gem",
    "Version": "0.1.0",
    "LinkType": "Dynamic",
    "Summary": "Our Gem",
    "Tags": ["Animation"],
    "IconPath": "OurGem.png",
    "EditorModule" : true
}
```

In the lumberyard_version\dev\Gems\OurGem\Code\wscript file, the uselib key must specify SUPERLIBRARY:

```python
def build(bld):
    bld.DefineGem(
        uselib = ['SUPERLIBRARY'],
        file_list  = ['our_gem.waf_files']
    )
```

### Step 4. Configure and Validate the Library Definition

Now you can test your definitions by using the Waf configure command, which processes third-party definitions. Before you begin, use the Project Configurator to ensure that the gem that has the third-party library is enabled for the game project. For more information, see Enabling Gems (p. 1064).

From the engine root path, run the configure command:

```
lmbr_waf configure
```

After the configure command completes successfully, the configured values appear in the Waf variant cache files in the BinTemp\c4che directory.

The entries are similar to the following ones in the variant cache file for the Microsoft Visual Studio 2017 profile configuration lumberyard_version\dev\BinTemp\c4che\win_x64_vs2017_profile_cache.py:

```python
... INCLUDES_SUPERLIBRARY = ['C:\\MyProj\\Lumberyard-1.16\\Gems\\MyGem\\External\\Include'] ...
```
Updating a Third-Party Library in a Gem

To update a third-party library that is defined in a gem, you don't have to create a new third-party definition file. Instead, you can update the existing definition file. As a best practice, place the third-party library files in a version-named subfolder. To maintain good dependency tracking, we recommend that you update the gem version when you update a third-party library.

Creating Third-Party Library Configuration Files for Waf

Lumberyard's Waf build system has the ability to incorporate third-party libraries. Waf's uselib mechanism can apply the proper dependency injection of a third-party library into any project or module. In the wscript file, the uselib attribute specifies all-caps identifiers that represent the third-party libraries. The identifiers are defined in the .json configuration files that can exist either in the global _WAF_\3rdParty directory or in the gem-specific 3rdParty directory. These configuration files provide details for the library, including include paths, library paths, and linkage information. The files also specify whether the library is shared, dynamic, or header-only.

Topics
- Key Features (p. 108)
- Supported Library Types (p. 110)
- Using Aliases (p. 113)
- Creating Entries for Multiple Related Libraries (p. 114)
- Third-Party Library Configuration File Attributes (p. 116)
- Special Reserved Words (p. 117)

Key Features

Key features of the uselib mechanism include centralized management and an abstraction layer that makes it easy to use the same libraries across Lumberyard projects and modules.

Centralized Management

Encapsulating the build details within a single file helps centralize the management of third-party libraries. Centralized management in a single file is also beneficial when changes like different include paths or libraries to the third-party library occur. These configuration files can be global to the entire engine, or gem-specific based on the gems that are enabled for the game.

Abstraction

The third-party identifiers provide an abstraction layer between the third-party library details and the consuming project or module. This layer makes it unnecessary to explicitly inject the paths for the library and include headers for each project or module that uses the library. Instead, the uselib mechanism with the third-party identifier provides the necessary injection of paths, libraries, and custom rules.
For example, assume you have two third-party libraries, StaticLibA and DynamicLibB, that you want to link into test project ProjectOne. Without the uselib system, the wscript file for ProjectOne would look like the following:

**Example**

**ProjectOne\wscript**

```python
def build(bld):
    bld.LumberyardSharedLibrary(
        target = 'ProjectOne',
        ...
        includes = ['3rdParty/StaticLibA/includes',
                    '3rdParty/DynamicLibB/includes'],
        win_stlibpath = ['3rdParty/StaticLibA/win32/lib'],
        darwin_stlibpath = ['3rdParty/StaticLibA/osx/lib'],
        ...
        win_libpath = ['3rdParty/DynamicLibB/win32/bin'],
        darwin_libpath = ['3rdParty/DynamicLibB/osx/bin'],
        ...
        lib = ['staticliba', 'dynamiclibb'],
        ...
        win_copy_dependent_files = ['3rdParty/DynamicLibB/win32/bin/dynamiclibb.dll'],
        darwin_copy_dependent_files = ['3rdParty/DynamicLibB/win32/bin/
libdynamiclibb.dylib'],
        ...
    )
```

Instead, you could use the following third-party configuration files for the two libraries, as in the following examples:

**Example**

**StaticLibA.json**

```json
{
    "name": "STATIC_LIB_A",
    "source": "3rdParty",
    "description": "My Static Library A",
    "includes": "includes",
    "platform": {
        "win_x64_vs2017": {
            "libpath": [
                "win32/lib"
            ],
            "lib": ["staticliba.lib"],
        },
        "darwin_x64": {
            "libpath": [
                "osx/lib"
            ],
            "lib": ["libstaticlibb.a"],
        }
    }
}
```

**Example**

**DynamicLibB.json**

```json
{
    "name": "DYNAMIC_LIB_B",
   ...
```
"source": "3rdParty",
"description": "My Dynamic Library B",
"includes": 
"platform": { 
  "win_x64_vs2017": { 
    "importlibpath": [ 
      "win32/lib"
    ],
    "import": [ 
      "dynamiclibb.lib"
    ],
    "shared": [ 
      "dynamiclibb.dll"
    ]
  },
  "darwin_x64": { 
    "importlibpath": [ 
      "osx/lib"
    ],
    "import": [ 
      "libdynamiclibb.a"
    ],
    "shared": [ 
      "libdynamiclibb.dylib"
    ]
  }
}

This now simplifies the wscript declaration for ProjectOne to the following:

Example

```
ProjectOne\wscript
def build(bld):
    bld.LumberyardSharedLibrary(
        target = 'ProjectOne',
        ...
        uselib = ['STATIC_LIB_A', 'DYNAMIC_LIB_B'],
        ...
    )
```

After you have created these third-party configuration files, you can apply the same simplification to any project or module in Lumberyard.

**Supported Library Types**

Third-party configuration files support the following library types:

- **Header Only** (p. 110)
- **Static** (p. 111)
- **Shared/Dynamic** (p. 112)

**Note**

For simplicity, the rest of this topic refers to shared/dynamic libraries as 'shared'.

**Header-only Libraries**

Header-only libraries provide only the include paths to the dependent project or module. The following example third-party library configuration file contains a definition for a header-only library:
Example

rapidxml.json

```json
{
  "name": "rapidxml",
  "source": "@3P:rapidxml@",
  "description": "Rapid XML header only library",
  "header_only": "True",
  "includes": [
    "include"
  ],
  "defines": [],
  "lib_required": "False"
}
```

This simple declaration specifies only the includes header path which is a subdirectory of the source directory. The source directory is defined by the alias pattern @3P:XXX@. For descriptions of the other attributes in this file, see Third-Party Library Configuration File Attributes (p. 116).

Static Libraries

Static third-party library configuration files declare header include paths, library include paths, and library names for inclusion in a dependent project or module. The following example third-party library configuration file contains a declaration for a simple static library.

Example

lz4.json

```json
{
  "name": "lz4",
  "source": "@3P:lz4@",
  "description": "LZ4 Compression Library",
  "includes": [
    "lib"
  ],
  "defines": [],
  "lib_required": "True",
  "shared_required": "False",
  "platform": {
    "win_x64_vs2017": {
      "libpath_debug": [
        "build/win_x64/vc140/debug"
      ],
      "libpath_release": [
        "build/win_x64/vc140/release"
      ],
      "lib": [
        "lz4.lib"
      ],
      "win_x64_vs2019": "@win_x64_vs2017",
      ...
    }
  }
```

Platform-Specific Information

In addition to the attributes described for header-only library definition, the preceding example includes platform-specific values. The platform attribute contains a dictionary of target platform sections. Each target platform must be specified here. If a particular target platform is not listed, this third-party library definition will not apply for that platform when built. Static libraries must specify the library
path (`libpath`) and the names of the library file (`lib`) against which to link for each platform. The configuration extension for the `libpath` attribute in the preceding example will be described later in this topic.

**Shared Libraries**

Shared libraries are more complex than static libraries. Shared libraries must declare an include path, import library path, shared library path, shared library file names, and optionally, program database (.pdb) files. The following example third-party library configuration file contains a definition for a shared library.

**Example**

`boostpython.json`

```json
{
  "name": "boostpython",
  "source": "@3P:boost@",
  "description": "Boost header library including python support libraries",
  "defines": [],
  "lib_required": "False",
  "platform": {
    "win_x64_vs2017": {
      "includes": ["."],
      "importlibpath": ["lib/windows/x64"],
      "sharedlibpath": ["lib/windows/x64"],
      "import_debug": ["boost_python-vc140-mt-gd-1_61.lib"],
      "import_release": ["boost_python-vc140-mt-1_61.lib"],
      "shared_debug": ["boost_python-vc140-mt-gd-1_61.dll"],
      "shared_release": ["boost_python-vc140-mt-1_61.dll"],
      "pdb_debug": ["boost_python-vc140-mt-gd-1_61.pdb"
      
    }
  },
}
```

The following table describes the attributes for shared library configuration files.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>importlibpath</td>
<td>Path to the import library that declares the exported symbols against which the dependent project or module links.</td>
</tr>
<tr>
<td>sharedlibpath</td>
<td>Path to the binaries of the shared library (the .dll file).</td>
</tr>
<tr>
<td>import</td>
<td>File names of the import libraries against which the dependent project or modules will link to get the exported symbols.</td>
</tr>
</tbody>
</table>
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared</td>
<td>File names of the binaries of the shared libraries.</td>
</tr>
<tr>
<td>pdb</td>
<td>File names of any pdbs that can be optionally copied over with the .dll file.</td>
</tr>
</tbody>
</table>

Besides providing the paths for header inclusion and linking, Lumberyard Waf also copies the shared library and PDB files to the path where the dependent project or module is built.

**Mixed Shared and Static Libraries**

In some cases, the third-party library is static on some platforms but shared on others. For cases like these, it is possible to declare mixed static and shared libraries for each platform.

**Using Aliases**

The following sections describe how to simplify the configuration file by using aliases for libraries, platforms, and paths.

**Aliasing Library Lists**

The same list of libraries might be needed in different platform sections of the configuration file. To avoid repetition, you can define and use a list alias that expands to a list of other values. This is helpful for cases in which you have multiple shared libraries that import libraries of the exact same name.

You define an alias in the *aliases* node under the root node in the configuration file. Add the name of the alias to the *aliases* node and a list of strings that the alias represents under the alias name.

In the following example, the *all_foo_lib_names* alias represents the libraries *foo_library_A* through *foo_library_D*. The library names must not have any prefixes or extensions.

**Note**

For some platforms, the file name to library name convention is different. For example, Linux-based libraries prefix the file name with *lib*, but Windows-based libraries do not.

In the *platform* section, the shared library definitions for *import*, *shared*, and *pdb* use the notation `${<ALIAS>}` to form the entries, where `<ALIAS>` is the name of the alias that was defined in the *aliases* section.
Aliasing Platform Declarations

Some different target platforms are binary compatible. In these cases, you do not have to declare a copy of an existing configuration section. Instead, you can use the platform attribute string alias `@<platform_name>` instead of another attribute dictionary.

In the following example, the `win_x64_vs2017` and `win_x64_vs2019` platforms are binary compatible, so the alias `@win_x64_vs2017` is used.

```
"platform": {
    "win_x64_vs2017": {
        "libpath_debug": ["build/win_x64/vc141/debug"],
        "libpath_release": ["build/win_x64/vc141/release"],
        "lib": ["foo.lib"
    ],
    "win_x64_vs2019": 
        "@win_x64_vs2017",
    ...
```

Aliasing Paths

Path aliasing for the `source` attribute is common within the configuration file. The alias value is denoted by `@XXXX@`, where the `XXXX` value resolves to an absolute path. The third-party configuration file recognizes two different path alias types:

- `@3P:YYYY@` – This alias refers to a third-party library that is managed by the Lumberyard Setup Assistant. These third-party libraries are globally available to the engine. Lumberyard Setup Assistant manages the location and version of these libraries independently from the build system. `YYYY` refers to the SDK identifier used by Lumberyard Setup Assistant and is different from the third-party `uselib` identifier used by Waf.
- `@GEM@` – This alias is valid only for gem-defined third-party libraries and refers to the base path of the gem from which the third-party library is defined. For information about putting third-party libraries in gems, see Adding Third-Party Libraries (p. 105).

Creating Entries for Multiple Related Libraries

Your project might have multiple libraries under a shared root source directory that are related but have separate purposes. This section shows how your configuration files can handle these more complex cases so that you can still take advantage of the `uselib` mechanism.

Build Configuration Filtering

Attributes can be constrained by build configuration. For example, a debug library and a release library might have the same name but exist in two different directories, as in the following example file structure:

```
/StaticLibA/
   /Include
       static_a.h
```
Using the Waf Build System

```plaintext
/lib
d/ebg
    static_a.lib
/release
    static_a.lib
```

The following example shows the configuration file declaration for this file structure. Notice that the `lib` attribute is specified once, but the `libpath` attribute is specified twice.

```plaintext
"include": ["Include"],
"libpath_debug": ["lib/debug"],
"libpath_release": ["lib/release"],
"lib": ["static_a.lib"]
```

If the libraries are in the same directory, but named differently, the file structure might look like this:

```plaintext
/StaticLibA/
/Include
    static_a.h
/lib
    static_aD.lib
    static_a.lib
```

In this case, the value for `libpath` in the configuration file is the same, but there are two values for `lib`:

```plaintext
"include": ["Include"],
"lib": ["lib"],
"lib_Debug": ["static_aD.lib"],
"lib_Release": ["static_a.lib"]
```

### Specifying Multiple Identifiers

A third-party library might contain a suite of libraries and be organized by specific categories according to need. In these cases, the configuration file can define more than one identifier. As long as the library files are under the same directory that is specified for the `source` attribute, the configuration can specify multiple third-party identifiers.

The following example configuration file defines two third-party identifiers that `uselib` will point to: `water_lib` and `air_lib`. The `lib` attribute is prefixed with the name of the third-party identifier. Both libraries are under the same elements library directory and exist individually as `water_lib.lib` and `air_lib.lib`. Both `water_lib` and `air_lib` require the same base `element_base.lib` library.

```plaintext
{
    "name": [
        "water_lib",
        "air_lib"
    ],
    "source": "elements",
    ...
    "platform": {
        "win_x64_vs2017": {
            "libpath": [
                "lib"
            ],
            "water_lib/lib": [
                "element_base.lib",
                "water_lib.lib"
            ],
            "air_lib/lib": [
```
"element_base.lib",
"air_lib.lib"
]

...}
%

...}

**Third-Party Library Configuration File Attributes**

The following attributes are used in third-party library definition files.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string/ [string]</td>
<td>Name of the third-party identifier. The value for the identifier set here is represented in Lumberyard in all caps. If the definition file defines multiple third-party identifiers, the name attribute contains a list of names.</td>
</tr>
<tr>
<td>source</td>
<td>string (aliasable)</td>
<td>Base directory upon which all library paths defined in the configuration file are based.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>Description for the third-party definition.</td>
</tr>
<tr>
<td>header_only</td>
<td>Boolean</td>
<td>Flag that indicates that the file specifies a header-only library definition.</td>
</tr>
<tr>
<td>includes</td>
<td>[string]</td>
<td>List of include paths to apply to any dependent project or module.</td>
</tr>
<tr>
<td>defines</td>
<td>[string]</td>
<td>List of additional defines to apply to any dependent project or module.</td>
</tr>
<tr>
<td>engine_configs</td>
<td>Boolean</td>
<td>When false, specifies that the library's debug configuration maps to the engine debug configuration and that release maps to profile, performance and release. The default is false. When true, accepts custom rules for each engine configuration. For example, if you want to use specific libraries for profile builds, you can specify values for libpath_profile and lib_profile. To copy specific binaries per configuration, you can also use attributes like copy_extra_debug, copy_extra_profile, and copy_extra_performance.</td>
</tr>
<tr>
<td>lib_required</td>
<td>Boolean</td>
<td>Flag that instructs the third-party configuration file parser whether to verify the existence of the declared static library files on disk.</td>
</tr>
<tr>
<td>shared_required</td>
<td>Boolean</td>
<td>Flag that instructs the third-party configuration file parser whether to verify the existence of the declared shared library files on disk.</td>
</tr>
<tr>
<td>suppress_warning</td>
<td>Boolean</td>
<td>Flag that disables warnings related to invalid third-party configurations. It is recommended that this flag be set to true for libraries that are optional.</td>
</tr>
<tr>
<td>platform</td>
<td>{ platform_def }</td>
<td>Dictionary of platform-specific settings (platform_def) that is keyed by the target platform name.</td>
</tr>
</tbody>
</table>
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkflags</td>
<td>[ string ]</td>
<td>List of linker flags to pass to the linker.</td>
</tr>
<tr>
<td>libpath</td>
<td>[ string ]</td>
<td>List of static library search paths to add to the consumers of this library. The paths are relative to the location specified by the source attribute.</td>
</tr>
<tr>
<td>lib</td>
<td>[ string ]</td>
<td>List of full static library file names (including file extensions) to add to the consumers of this library. The system validates the existence of the files by searching the locations defined in the libpath attribute.</td>
</tr>
<tr>
<td>sharedlibpath</td>
<td>[ string ]</td>
<td>List of paths that contain the shared libraries for this uselib definition. The paths are relative to the location specified by the source attribute.</td>
</tr>
<tr>
<td>shared</td>
<td>[ string ]</td>
<td>List of the shared library file names for this uselib definition. These shared libraries are copied to the output directory of the target of the consumer module.</td>
</tr>
<tr>
<td>importlibpath</td>
<td>[ string ]</td>
<td>Applies to Windows-based platforms only. A list of paths that contain import libraries for linking DLLs for this uselib definition. The paths are relative to the location specified by the source attribute.</td>
</tr>
<tr>
<td>import</td>
<td>[ string ]</td>
<td>Applies to Windows-based platforms only. A list of import library file names that represent this uselib definition. These libraries are used in the linker command.</td>
</tr>
<tr>
<td>frameworkpath</td>
<td>[ string ]</td>
<td>Applies to OSx-based platforms only. A list of framework paths that contain frameworks for this uselib definition. The paths are relative to the location specified by the source attribute.</td>
</tr>
<tr>
<td>framework</td>
<td>[ string ]</td>
<td>Applies to OSx-based platforms only. A list of frameworks that represent this uselib definition.</td>
</tr>
<tr>
<td>copy_extra</td>
<td>[ string ]</td>
<td>List of additional files to copy from the source to the target directory where the executables that consume this uselib definition reside. Each item in the list is a colon-delimited source-destination pair in the format <code>&lt;source&gt;:&lt;destination&gt;</code>. <code>&lt;source&gt;</code> specifies the source directory that contains the files relative to the location specified by the source attribute. <code>&lt;destination&gt;</code> is the directory to which the files are copied relative to the destination directory of the consuming target executable.</td>
</tr>
</tbody>
</table>

### Special Reserved Words

The following special reserved words are used in third-party library configuration files.

<table>
<thead>
<tr>
<th>Reserved Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@GEM@</td>
<td>Alias that represents the base directory of the gem. This is used for third-party configurations that reside within the gem.</td>
</tr>
<tr>
<td>%LIBPATH(&lt;libname&gt;)</td>
<td>Specifies a macro that resolves to the full path of <code>&lt;libname&gt;</code>. <code>&lt;libname&gt;</code> must be defined within the scope of the platform for which the macro is used.</td>
</tr>
</tbody>
</table>
Reserved Word | Description
--- | ---
| | is used. For example, if there is a library called libmath.lib whose absolute path is C:\libs, %LIBPATH(libmath.lib) resolves to C:\libs\libmath.lib. This is useful for linker flags like --force-load <libfullpath> that can be added as a link flag parameter. In the following example, libtomcrypt requires a force load on its library through its linker flags. The definition that achieves this is as follows:

```json
"linkflags": [
  "-force_load",
  "%LIBPATH(libtomcrypt.a)"
]
```

Waf Extensions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Compiling with IncrediBuild

Waf supports IncrediBuild 8.0.1 or later, and allows for distributed network builds for compiling larger projects.

You must have the appropriate package for your operating system:

- **Windows** or **Android** – IncrediBuild for Make and Build or Accelerated Dev Tools

To verify which package is configured for your machine, run the following command (located in C:\Program Files (x86)\Xoreax\IncrediBuild): xgConsole.exe /QUERYLICENSE

The following is output:

```plaintext
> xgConsole.exe /QUERYLICENSE
License details:
----------------
Registered to: My Game Company
Up to XX Agents allowed
Maintenance expires on XX/XX/XXXX
Packages installed:
-------------------
- IncrediBuild for Make & Build Tools
```

<table>
<thead>
<tr>
<th>To do this</th>
<th>Run this at a command line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable or disable IncrediBuild builds</td>
<td>use_incredibuild</td>
</tr>
<tr>
<td>Adjust the maximum number of parallel tasks</td>
<td>incredibuild_max_cores</td>
</tr>
</tbody>
</table>
Using the Waf Build System

To do this

Determine which IncrediBuild package is configured for your machine

Run this at a command line

xgConsole.exe /QUERYLICENSE

Waf requires certain packages and the Windows registry key settings below to run IncrediBuild. Run lmb\_\_waf\_\_\_waf.bat in Administrator mode to edit the registry.

Modify the settings in the Windows registry under the following key:

HKEY\_LOCAL\_MACHINE\Software\Wow6432Node\Xoreax\Incredibuild\Builder

Registry Settings

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxConcurrentPDBs</td>
<td>0</td>
<td>Controls how many files can be processed in parallel. This optimization is also useful for MSBuild.</td>
</tr>
</tbody>
</table>

To enable IncrediBuild

1. Open the user\_settings\_options file located in //_WAF_._
2. In the user\_settings\_options file, under [IncrediBuild Options], do the following:
   * Set the use\_incrdibuild flag to True.
3. Save your changes.

Note

Installing, reinstalling, or upgrading Visual Studio may cause the Incredibuild Agent to lose its settings or require an update. After modifying Visual Studio, be sure to confirm your Incredibuild Agent is active.

Compiling with QT

Waf supports compiling QT5 .moc Meta-Object-Compiler files. To enable or disable compiling of particular files, add the qt5 feature to your Waf Module (wscript) file and then add the list of files to be compiled.

The following example shows a Waf Module (wscript) file:

```python
# wscript relative path
QT\_TO\_MOC\_FILES = [ 
  'MyQTFile.h',
  'MyOtherQTFile.h',
  ...
]
def build(bld):
  bld.CryPlugin(
    target = 'MyQTPlugin',
    vs_filter = 'Plugins',
    file_list = 'file_list.waf_files',
    features = ['qt5'], # add the QT5 moc feature to this Waf module
  )
```

Version 1.28

119
Using Waf

Adding a Game Project

The simplest and recommended method to add a game project to the Lumberyard Waf build system is to use the Project Configurator. The Project Configurator is a standalone application for telling the Waf build system which game projects and assets to include in a game build. For more information, see Creating Lumberyard projects (p. 43).

You can also add a game project with the following steps:

- Create the project definition
- Create a game module
- Update the user settings to include the game

**Note**
You can build your game project by creating a game project first (see steps below) and then creating a spec for just the game (no modules, just basic spec values):

```
{
    "description": "Configuration to build the My Game",
    "visual_studio_name": "My Game"
}
```

When the project is properly defined and all source files are in the correct locations, you can set the enabled_game_projects value in the user_settings.options file. Configuring this value limits the Visual Studio solution to the launcher projects and your game project.

**Topics**
- Creating the Project Definition (p. 120)
- Creating a Game Module (p. 121)
- Updating the User Settings (p. 122)
the game-specific assets. You can define any number of game projects in this file and you can configure which ones to build.

**To create the project definition**

1. Navigate to the SDK root and locate the Code folder and project directory. Typically your game code folder should reside under these locations.
2. Determine the name for your project. For this example, use **My Game**.
3. Add the definitions for the new game project to the `project.json` file (located in the game project folder under the `lumberyard_version\dev` directory). For this example, add **My Game** to the SDK:

```
"project_name" : "SamplesProject",
"product_name" : "Samples Project",
"executable_name" : "SamplesProjectLauncher",
"code_folder" : "Code/SamplesProject",
"modules" : ["SamplesProject"],
"project_id": "{D882E365-54D6-586E-BD78-2650F3057D49}",
"sys_game_folder" : "SamplesProject",
"sys_dll_game" : "SamplesProject",
```

**Creating a Game Module**

You can create a game module after setting the game project definition. Game modules include wscript files, source files, and a waf_files configuration file. You must create separate folders for the game source code and for the resources. Both should reside under the `code_folder` specified earlier. For this example you create folders called **GameSource** and **Resources** under the `Code/MyGame` directory.

**Create a wscript file**

Because Waf searches for and discovers wscript files recursively through other wscript files, you must include a simple wscript file in the `Code/MyGame` folder that recurses to the `GameSource` folder.

Create a file with the following:

```python
SUBFOLDERS = ['GameSource']

def build(bld):
    bld.recurse(SUBFOLDERS)
```

Next you must create the source code in the **GameSource** folder. Include in this folder all of your source files and the corresponding Waf source file configuration (for example, **MyGame.waf_files**) to include your game files.

Create a wscript in the **GameSource** folder to define the build configuration for your game:

```python
def build(bld):
    bld.CryEngineModule(
        target = 'MyGame',
        vs_filter = 'Game/MyGame',
        file_list = 'MyGame.waf_files',
        pch = 'StdAfx.cpp',
    )
```
Create source files

All game projects first need a source file. If you intend to use pre-compiled headers you must create standard `StdAfx.h` and `StdAfx.cpp` files. For this example you create a single C++ file and a corresponding header file (`MyGameMain.cpp` and `MyGameMain.h`).

Create a waf_files configuration file

You use the waf_files configuration file to include the source files into the game module. For this example you create a file called `MyGame.waf_files` and specify it for the project. This file includes the four files you created from the previous step.

Create a waf_files configuration file called `MyGame.waf_files` with the following:

```json
{
  "auto": {
    "Source Files": ["MyGameMain.cpp"],
    "Header Files": ["MyGameMain.h"
  },
  "none": {"Root": [
    "StdAfx.h",
    "StdAfx.cpp"
  ]
}
}
```

Updating the User Settings

The final step is to update enabled_game_projects to include or exclusively set the new game project. You can do this one of the following ways:

- In a text editor, edit the `user_settings.options` file (in the `lumberyard_version\dev\WAF_\` directory) to set the value for the `enabled_game_projects`. The following example sets `MyGame` as the only game project generated. You can use a comma-separated list to include multiple game projects in the final solution.

  ```
  [Game Projects]
  enabled_game_projects = MyGame
  ```

- Update game projects with the Lumberyard Waf GUI.

  To run the GUI, in a command line window, navigate to the `lumberyard_version\dev\` directory and enter the following command:

  ```
  lmbr_waf show_option_dialog
  ```

  Click **Game Projects** in the Lumberyard Waf window, and select your new project. You can select more than one project.
• Build the project with the following command for your version of Visual Studio. Use `--enabled-game-projects=MyGame` to override every build command. This does not include the project in the generated solution, but it sets specific game projects to build during the build commands.

```bash
lmbr_waf build_win_x64_vs2017_debug -p game_and_engine --enabled-game-projects=MyGame
```

**Adding a Spec**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Waf spec system provides a template to create Visual Studio solutions and describes a build filter that determines which modules to build for particular platforms and configurations. The nature of the generic Waf build system is to be all projects that are defined through the wscript system, which acts recursively on the root directory structure. If no spec is specified when you execute a build or clean command, the Waf build system system attempts to build all modules that are supported by the selected target platform and configuration. The platform and configuration support is defined in each of the module's wscript definitions. For more information, see Adding a Build Module (p. 125).

Project spec files are a collection of modules and definitions for a specific build pipeline. These files are useful for including existing modules or adding new ones as part of the build dependencies for your game project.

When you build in debug or profile configurations and their _dedicated counterparts, a spec file is not required. This is because these two configurations build out of Lumberyard as modular shared components. In performance and release configurations, however, all the modules that are marked as CryEngineModules are built monolithically, which means that they are built into a single executable. This causes problems with similar modules that support the same platform and configuration. Currently, the spec file is required for this scenario in order to target specific modules to build into the monolithic .exe files.

Adding a project requires these steps:

• Creating a New Project Spec JSON File (p. 123)
• Adding the Spec File to the Visual Studio Solution Generator (p. 124)
• Building the Spec (p. 125)

**Topics**

• Creating a New Project Spec JSON File (p. 123)
• Adding the Spec File to the Visual Studio Solution Generator (p. 124)
• Building the Spec (p. 125)

**Creating a New Project Spec JSON File**

In the following example a spec file called `my_game` includes the game engine modules as a base as well as custom modules for Windows. The spec file also sets a custom #define for Windows builds.

You need to configure the values for the modules that you want to include in the spec file (and optionally the target platform and configuration). The spec file can isolate target_platform modules for multiplatform builds.

Create a spec file called `my-game.json` with the following:

```json
{
```

Version 1.28

123
Using the Waf Build System

The spec files are located in the `lumberyard_version\dev\WAF\specs` directory and have the `.json` file extension. For more information about Waf spec files, see Waf Spec Files (*.json) (p. 74).

<table>
<thead>
<tr>
<th>Spec</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all.json</td>
<td>Configuration for all targets.</td>
</tr>
<tr>
<td>dedicated_server.json</td>
<td>Configuration to build dedicated servers for the enabled projects.</td>
</tr>
<tr>
<td>external_sdks.json</td>
<td>Configuration to build externally distributed binary-only libraries.</td>
</tr>
<tr>
<td>game_and_engine.json</td>
<td>Configuration to build the engine and game projects.</td>
</tr>
<tr>
<td>pipeline.json</td>
<td>Configuration to build Pipeline Only for building Resource Compiler, and also Maya, 3ds Max, and Photoshop plugins if Visual Studio 2010 and 2012 are installed. Build only in Profile or Debug mode (Release mode is only for the 3ds Max plugin)</td>
</tr>
<tr>
<td>resource_compiler.json</td>
<td>Configuration to build only the Resource Compiler.</td>
</tr>
<tr>
<td>shadercachegen.json</td>
<td>Configuration to build only the shadercache generator.</td>
</tr>
<tr>
<td>tools.json</td>
<td>Configuration to build nonessential tools.</td>
</tr>
</tbody>
</table>

Adding the Spec File to the Visual Studio Solution Generator

Adding the spec file to the Visual Studio solution is optional.

To add the spec file to the Visual Studio solution

1. In a text editor, edit the `specs_to_include_in_project_generation` value in the `user_settings.options` file to add your spec file to the Visual Studio solution:

   ```
   [Visual Studio Project Generator]
   generate_vs_projects_automatically = True
   visual_studio_solution_name = LumberyardSDK
   visual_studio_solution_folder = Solutions
   specs_to_include_in_project_generation = MySpec1, MySpec2, MySpec3
   ```

2. Enter the following command to regenerate the Visual Studio solution:
Building the Spec

After saving the new spec, do one of the following:

- Build the spec using Visual Studio (if you followed the steps above to add the spec to Visual Studio).
- Build the spec from the command line by entering the following command for your version of Visual Studio:

```
lmbr_waf build_win_x64_vs2017_profile -p MySpec
```

The `build` command builds the game project specified in the `user_settings`, even if the module is not defined in the spec. The exception is if the option `disable_game_projects` is set to `True`.

Adding a Build Module

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create a custom build module in the Lumberyard Waf build system. You can use predefined build modules to add any shared library or plugin into the Lumberyard engine SDK.

The default Waf system defines modules and methods that will take various keywords into Waf commands to build applications and shared and static libraries as well as serving as a project container for files. There `cryengine_modules.py` file defines functions that wrap these modules with additional keywords and logic to extend the behavior of standard Waf into a system that supports the requirements of Lumberyard. In addition to providing standard Waf build functionality, the functions in the various modules add support for precompiled headers (`pch`), content file support (`.waf_files`), monolithic build capability, uber file support, and Microsoft Visual Studio (msvs) generation.

Creating a module requires the following steps:

1. Create the source folder and script
2. Create a basic `wscript` module
3. Create the `.waf_files` content file
4. Specify additional include paths and external library linking
5. Add a project dependency

Creating a New Module

You can create and add the following types of modules to the Lumberyard Waf build system:

<table>
<thead>
<tr>
<th>Build Module</th>
<th>Description</th>
<th>Project Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CryEngineModule</td>
<td>Modules that are dynamically loaded at runtime as part of the lumberyard engine module system. For Performance and Release configurations, all projects that are built using</td>
<td>Shared Library (Non-Release), Static Library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Performance (Performance, Release)</td>
</tr>
</tbody>
</table>

Version 1.28

125
## Using the Waf Build System

<table>
<thead>
<tr>
<th>Build Module</th>
<th>Description</th>
<th>Project Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>These modules are included monolithically to the final build output. If the libraries are not linked in, the source from these modules is included in the build.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For Debug and Profile configurations, these modules are built as shared libraries. For the Windows platform, versioning information is injected as defined in the waf_branch_spec.py file located in the root folder. As such, a Windows resource (.rc) file as needed as part of the waf_files content.</td>
<td></td>
</tr>
<tr>
<td>CryEngineSharedLibrary</td>
<td>Used to define a shared library that any other module can use inside Waf. Provided they are located in the same directory path as the dependent project, these modules are included as a dependency to other modules by use of the <code>use</code> keyword.</td>
<td>Shared Library</td>
</tr>
<tr>
<td>CryEngineStaticLibrary</td>
<td>Used to define a static library that can be used by any other module inside Waf. Provided they are located in the same folder path as the dependent project, these modules are included as a dependency to other modules by use of the <code>use</code> keyword.</td>
<td>Static Library</td>
</tr>
<tr>
<td>CryLauncher</td>
<td>Used to define the build definition for launchers, which are created for each game project defined per supported platform. All supported launchers that can be generated based on availability against the current platform are located in the \Code\Launcher subfolder. If an additional platform is included, a new launcher project would be added in this subfolder and use the CryLauncher build module.</td>
<td>Executable</td>
</tr>
<tr>
<td>CryDedicatedServer</td>
<td>Similar to the CryLauncher module, except used for dedicated server projects.</td>
<td>Executable</td>
</tr>
<tr>
<td><strong>Build Module</strong></td>
<td><strong>Description</strong></td>
<td><strong>Project Type</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>CryConsoleApplication</strong></td>
<td>Used to build console applications. On the Windows platform, it builds a console application instead of a Windows application.</td>
<td>Executable</td>
</tr>
<tr>
<td><strong>CryBuildUtility</strong></td>
<td>Used to define build utility projects, such as AZCodeGenerator. Build utilities are separated into a build_utilities group that are built before the regular build group.</td>
<td>Executable</td>
</tr>
<tr>
<td><strong>CryFileContainer</strong></td>
<td>Used to set a file container for projects.</td>
<td>None</td>
</tr>
<tr>
<td><strong>CryEditor</strong></td>
<td>Used by Lumberyard Editor projects.</td>
<td>Executable</td>
</tr>
<tr>
<td><strong>LumberyardApp</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CryEditorUiQt</strong></td>
<td>Used by the CryEditorUI_QT plugin.</td>
<td></td>
</tr>
<tr>
<td><strong>CryPlugin</strong></td>
<td>Used by Lumberyard Editor plugin projects. It is automatically placed in the EditorPlugins subfolder and automatically loaded by Lumberyard Editor at runtime.</td>
<td>Shared Library</td>
</tr>
<tr>
<td><strong>CryStandAlonePlugin</strong></td>
<td>Used by Lumberyard Editor plugin projects. The difference between this module and CryPlugin is that it does not import any SANDBOX or EDITOR_COMMON imports, RTTI is enabled, and nodefaultlib:/ is set to libcmt.</td>
<td>Shared Library</td>
</tr>
<tr>
<td><strong>CryPluginModule</strong></td>
<td>Used to define shared libraries that can be used by a Lumberyard Editor plugin. Plugins that need to link to a Cryengine plugin module use the use feature of Waf.</td>
<td>Shared Library</td>
</tr>
<tr>
<td><strong>CryResourceCompiler</strong></td>
<td>Used by the Resource Compiler to implicitly set the target name to rc and the subfolder to rc under the configure output folder.</td>
<td>Executable</td>
</tr>
</tbody>
</table>
Using the Waf Build System

Build Module | Description | Project Type
--- | --- | ---
CryResourceCompiler | Used by the Resource Compiler to implicitly set the target name to rc and the subfolder to rc under the \configure output folder. | Shared Library
CryPipelineModule | Used to define pipeline modules such as for the 3ds Max and Maya exporters. | Custom
CryQtApplication | Used to define Qt 5 applications that can be launched by Lumberyard Editor, such as the Asset Processor. | Executable
CryQtConsoleApplication | Used to define Qt 5 console applications that can be launched by Lumberyard Editor, such as the Asset Processor batch file. | Executable

In this topic's example you create a CryEngineModule.

**Build Module Keywords**

The following describes the general keywords that are supported by the build modules. The listed targetable keywords can be specific to a platform or a configuration. The keyword by itself is used for all supported platforms and configurations, but if you need keywords that are specific to a platform or configuration, you must include the name of the platform or configuration in the name.

**Other things to consider:**

- The general pattern for platform plus configuration-specific values is `<platform>_<configuration>_<keyword>`.
- The general pattern for platform-specific values is `<platform>_<keyword>`
- The general pattern for configuration-specific values is `<configuration>_<keyword>`

You can use the following keyword macros to reduce the verboseness of `wscript` files:

**<platform>_nddebug_<keyword>**

This macro eliminates the need to repeatedly specify certain non-debug flags. Lumberyard has one debug configuration and three non-debug configurations.

| Keyword | Description | Targetable?
--- | --- | ---
| target | Name of the target project. | No
| platforms | The list of platforms to restrict this module to. If not specified, then defaults to all, which assumes all supported target platforms on the current host. | No

Version 1.28
128
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Targetable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>configurations</td>
<td>The list of configurations to restrict this module to. If not specified, then defaults to all. In addition to the standard configurations (debug, profile, and release), configurations can be specific to a particular platform. This is done by appending the platform name with a colon separator in front of the configuration. For example, if a module supports only debug and profile for the iOS platform, then the configuration list would include the values <code>ios:debug</code> and <code>ios:profile</code>.</td>
<td></td>
</tr>
<tr>
<td>file_list</td>
<td>The <code>.waf_files</code> JSON file that contains the file list definition for the project.</td>
<td>Yes</td>
</tr>
<tr>
<td>pch</td>
<td>The name of the precompiled header. If present, then precompiled headers are enabled.</td>
<td></td>
</tr>
<tr>
<td>use</td>
<td>Additional projects to link as a use dependency.</td>
<td>Yes</td>
</tr>
<tr>
<td>uselib</td>
<td>Additional libraries to use.</td>
<td>Yes</td>
</tr>
<tr>
<td>defines</td>
<td>Additional preprocessor defines for the project.</td>
<td>Yes</td>
</tr>
<tr>
<td>includes</td>
<td>Additional include paths.</td>
<td>Yes</td>
</tr>
<tr>
<td>cflags</td>
<td>Additional C flags.</td>
<td>Yes</td>
</tr>
<tr>
<td>cxxflags</td>
<td>Additional C++ flags.</td>
<td>Yes</td>
</tr>
<tr>
<td>lib</td>
<td>Additional libraries to link to.</td>
<td>Yes</td>
</tr>
<tr>
<td>libpath</td>
<td>Additional library include path.</td>
<td>Yes</td>
</tr>
<tr>
<td>stlib</td>
<td>Boolean flag that indicates a static library module.</td>
<td>Yes</td>
</tr>
<tr>
<td>stlibpath</td>
<td>Lib path for static libs (generally the same for any lib).</td>
<td>Yes</td>
</tr>
<tr>
<td>linkflags</td>
<td>Additional link flags during the linker phase.</td>
<td>Yes</td>
</tr>
<tr>
<td>export_definition</td>
<td>Export definition filename (.def file).</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Creating a Basic Wscript Module

The `wscript` file specifies the name of the module (`target`), `.waf_files` content file (`file_list`), Visual Studio filter (`vs_filter`), and precompiled headers (`pch`).

Create a `wscript` module with the following:

```python
def build(bld):
    bld.CryEngineModule(
        target      = 'MyEngineModule',
        vs_filter   = 'LyEngine',
        file_list   = 'myenginemodule.waf_files',
        pch         = 'StdAfx.cpp'
    )
```

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Targetable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>features</td>
<td>Any additional features to tag this project to.</td>
<td>Yes</td>
</tr>
<tr>
<td>output_file_name</td>
<td>An output file name used to override the default output file based on the target.</td>
<td>Yes</td>
</tr>
<tr>
<td>framework</td>
<td>Additional frameworks (Darwin).</td>
<td>No</td>
</tr>
<tr>
<td>frameworkpath</td>
<td>Additional framework paths (darwin).</td>
<td>No</td>
</tr>
<tr>
<td>export_defines</td>
<td>Additional preprocessor defines that are added to any module that uses the current module as a project dependency.</td>
<td>No</td>
</tr>
<tr>
<td>export_includes</td>
<td>Additional library include paths that are added to any module that uses the current module as a project dependency.</td>
<td>No</td>
</tr>
<tr>
<td>additional_settings</td>
<td>Additional settings added for specific files.</td>
<td>Yes</td>
</tr>
<tr>
<td>meta_includes</td>
<td>Meta includes for WinRT.</td>
<td>Yes</td>
</tr>
<tr>
<td>files</td>
<td>Another way to pass in files for processing a build project.</td>
<td>Yes</td>
</tr>
<tr>
<td>winres_includes</td>
<td>Additional include paths for the winres compiler.</td>
<td>No</td>
</tr>
<tr>
<td>winres_defines</td>
<td>Additional defines for the winres compiler.</td>
<td>No</td>
</tr>
<tr>
<td>enable_rtti</td>
<td>Flag to enable rtti settings for a project.</td>
<td>Yes</td>
</tr>
<tr>
<td>rpath</td>
<td>Additional relative library paths (Darwin).</td>
<td>No</td>
</tr>
</tbody>
</table>

Version 1.28
130
In order for the Lumberyard Waf build system to pick up the new folder and script, you must add the new folder to the list of subfolders to recurse. Because you are adding this project under `lumberyard_version/Code/CryEngine/MyEngineModule`, you need to update the wscript located in the parent `lumberyard_version/Code/CryEngine` folder.

Update the wscript located in the `lumberyard_version/Code/CryEngine` folder with the following:

```python
SUBFOLDERS = ['CryInput', 'Cry3DEngine', ...
              'MyEngineModule',
              ]

def build(bld):
    # Recursive into all sub projects
    bld.recurse(SUBFOLDERS)

Creating the .waf_files Content File

In the example wscript, you specified a file called `myenginemodule.waf_files` as the project content file. The project content file can be one of the following:

- A single file that defines the source files for the project
- A list of files that define the source files for the project
- Platform/configuration, where certain files are included only for a particular platform (for example, console-specific files)

The following `myenginemodule.waf_files` example demonstrates a simple module with six files:

```json
{
    "NoUberFile":
    {
        "Root":
        [
            "StdAfx.cpp",
            "StdAfx.h"
        ],
    },
    "myenginemodule_uber_0.cpp":
    {
        "Root":
        [
            "myenginecore.cpp",
            "myenginecore.h",
            "myengineextras.cpp",
            "myengineextras.h"
        ]
    }
}
```

Specifying Additional Include Paths and External Library Linking

To configure the module to link to external modules, you need to update the wscript to specify the include path and link related project settings flags such as includes, lib, libpath, and linkflags.

In this example, you add the following to your module:

1. Google mock libraries for Win x64
2. Preprocessor DEFINE called `USE_GMOCK` to inject into the compile based on the platform Win x64
3. Link-time code generation flag to enable instrumentation (\`/LTCG:PGOPTIMIZE\`)

Add the following to your wscript module:

```python
def build(bld):
    bld.CryEngineModule(
        target          = 'MyEngineModule',
        vs_filter       = 'LyEngine',
        file_list       = 'myenginemodule.waf_files',
        pch             = 'StdAfx.cpp',
        win_includes    = [Path('Code/SDKs/GoogleMock/include')],
        win_lib         = ['gmock'],
        win_linkflags   = ['/LTCG:PGOPTIMIZE'],
        win_defines     = ['USE_GMOCK'],
        win_x64_debug_libpath       = [Path('Code/SDKs/GoogleMock/bin/x64/Debug')],
        win_x64_profile_libpath     = [Path('Code/SDKs/GoogleMock/bin/x64/Release')],
        win_x64_performance_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/Release')],
        win_x64_release_libpath     = [Path('Code/SDKs/GoogleMock/bin/x64/Release')]
    )
```

**Note**

The following are duplicated to cover all possible configurations that you specified in the `waf_branch_spec`: `win_x64_profile_libpath`, `win_x64_profile_performance`, and `win_x64_release_libpath`.

### Adding and Linking to a Project Dependency

If you want to link to another module that is built within the system, you can use the `use` parameter for the build.

Update your wscript module to link to the CryPerforce module:

```python
def build(bld):
    bld.CryEngineModule(
        target          = 'MyEngineModule',
        vs_filter       = 'LyEngine',
        file_list       = 'myenginemodule.waf_files',
        pch             = 'StdAfx.cpp',
        use             = ['CryPerforce'],
        win_includes    = [Path('Code/SDKs/GoogleMock/include')],
        win_lib         = ['gmock'],
        win_defines     = ['USE_GMOCK'],
        win_x64_debug_libpath       = [Path('Code/SDKs/GoogleMock/bin/x64/Debug')],
        win_x64_profile_libpath     = [Path('Code/SDKs/GoogleMock/bin/x64/Release')],
        win_x64_performance_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/Release')],
        win_x64_release_libpath     = [Path('Code/SDKs/GoogleMock/bin/x64/Release')]
    )
```

### Launching the Game Project

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Building your game project produces an `.exe` file in Lumberyard's file system.

To launch your project, open the executable file. The directory and file name are named according to the following conventions:

```
/dev/Bin64compiler.build mode/executable name.exe
```

Where:

- `compiler`
- `build mode` (p. 61)
  - `Debug` – Built in debug mode.
  - `(None)` – Built in profile mode.
  - `Performance` – Built in performance mode.
  - `Release` – Built in release mode.
- `executable name` – The `executable_name` specified in your `project.json` (p. 69) file. The file name typically mirrors the project name.

For example, if your project was configured with the following:

- Compiler: Visual Studio 2017
- Build mode: `debug`
- Executable name: `SampleProjectLauncher`

Then your directory and file name would be the following:

```
{lumberyard_version}/dev/Bin64vc141.Debug/SampleProjectLauncher.exe
```

### Adding User Settings to Waf

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add a new user setting to the `default_settings.json` file in the `Waf` folder located at the root. Use the standards established in this file and customize as needed. After you have added a user setting, you need to add a minimum of three utility functions for the GUI and console to validate your new setting.

To define utility functions, add the following to `default_settings.py`:

- **Getter** – Retrieves the value of your new setting and performs necessary transformations
- **Validator** (optional) – Validates new values
- **Hinter** (optional) – Tells GUI the available options

See the sections below for more information about these functions.

You can also add these functions to any new `.py` file as long as you add the module during build and configure. Be sure to load the file using the following command:
Using the Waf Build System

Getter Function

Waf calls the getter function to retrieve the value of your new setting and perform any necessary transformations.

Follow these guidelines:

• Implement the \texttt{@register\_attribute\_callback} function for your type.
• Use the same name for your function as your property name that's defined in the \texttt{default\_settings} file. For example, if your property name is called \texttt{my\_setting}, the function must be called \texttt{my\_setting()}.
• Choose attribute names that are unlikely to conflict.

In the example below of a getter/setter function, the current value is the input and the return value is the value with any validation and transformations applied. We expect a list of comma-separated values. The first half of the function returns the value quickly and the second half is where Waf runs in interactive or GUI mode.

```python
@register_attribute_callback
def enabled_game_projects(ctx, section_name, option_name, value):
    """Configure all Game Projects enabled by user""
    if ctx.options.execsolution or not ctx.is_option_true('ask_for_user_input'):
        return value
    if LOADED_OPTIONS.get('enabled_game_projects', 'False') == 'False':
        return ''

    info_str = ['Specify which game projects to include when compiling and generating project files.]
    info_str.append('Comma separated list of Game names, from the project.json root (SamplesProject, MultiplayerProject) for example')
    # GUI
    if not ctx.is_option_true('console_mode'):
        return ctx.gui_get_attribute(section_name, option_name, value, '\n'.join(info_str))
    # Console
    info_str.append("\nQuick option(s) (separate by comma):")
    project_list = ctx.game_projects()
    project_list.sort()
    for idx, project in enumerate(project_list):
        output = '   %d: %s: ' % (idx, project)
        while len(output) < 25:
            output += ' '  
        output += ctx.get_launcher_product_name(project)
        info_str.append(output)
    info_str.append("(Press ENTER to keep the current default value shown in [])")
    Logs.info(''.join(info_str))
    while True:
        projects = _get_string_value(ctx, 'Comma separated project list', value)
        projects_input_list = projects.replace(' ', '').split(',')
        # Replace quick options
        options_valid = True
        for proj_idx, proj_name in enumerate(projects_input_list):
            if proj_name.isdigit():
                option_idx = int(proj_name)
                try:
                    projects_input_list[proj_idx] = project_list[option_idx]
                except:
                    options_valid = False
        if not options_valid:
            Logs.error('Quick option(s) invalid. Try again or consult the manual for more information.')
        else:
            return projects_input_list
```
Logs.warn('[WARNING] - Invalid option: "%s" % option_idx)
options_valid = False
if not options_valid:
    continue
projects_enabled = ','.join(projects_input_list)
(res, warning, error) = ATTRIBUTE_VERIFICATION_CALLBACKS['verify_enabled_game_projects'](ctx, option_name, projects_enabled)
if error:
    Logs.warn(error)
    continue
return projects_enabled

In the example below, the function is simpler because it's a simple string entry and there are no enumerations like bool and no validation.

@register_attribute_callback
def out_folder_linux64(ctx, section_name, option_name, value):
    """ Configure output folder for linux x64 ""
    if not _is_user_input_allowed(ctx, option_name, value):
        Logs.info('User Input disabled.
Using default value "%s" for option: "%s" % (value, option_name))
        return value
    # GUI / console mode
    if not ctx.is_option_true('console_mode'):
        return ctx.gui_get_attribute(section_name, option_name, value)
    return _get_string_value(ctx, 'Linux x64 Output Folder', value)

Validator Function

Waf only requires the getter function; however, to validate input or provide the GUI with more than raw strings, you'll need to implement other functions like the validator.

Follow these guidelines:

- Implement the @register_verify_attribute_callback function and name it verify(_your_option_name_).
- Pass into the function the value parameter, which is the current raw value.
- Return a tuple of Bool, String, ErrorString. The first bool specifies whether or not validation is okay.

In the example below of a validator function, we make sure not to trigger the duplicate check (for example with a list like "SamplesProject,SamplesProject,SamplesProject") or provide a list that won't be accepted (for example with a list like "ASDJASUIDIASJDA").

@register_verify_attribute_callback
def verify_enabled_game_projects(ctx, option_name, value):
    """ Configure all Game Projects which should be included in Visual Studio ""
    if not value:
        return True, "", "" # its okay to have no game project
    if (len(value) == 0):
        return True, "", ""
    if (value[0] == ' ' and len(value) == 1):
        return True, "", ""
Hinter Function

Waf uses the optional hinter function to provide the GUI with a list of available options. For example, you might want to use the hinter function if you have a string list that can have multiple or single values that must be specific (enums).

Follow these guidelines:

- Implement the `@register_hint_attribute_callback` function and name it `hint_(your_option_name)`.
- Ignore the value parameter passed, which is the current value.
- Return a tuple of display value list, actual value list, help text list, multi or single. All three input lists should be the same length. The values in these lists are what's displayed in the GUI, the values to set if selected, and the text to display as extra information for an option, respectively.

The example below is for a hinter function.

```python
@register_hint_attribute_callback
def hint_enabled_game_projects(ctx, section_name, option_name, value):
    """ Hint list of specs for projection generation """
    project_list = ctx.game_projects()
    project_list.sort()
    desc_list = []
    for gameproj in project_list:
        desc_list.append(ctx.get_launcher_product_name(gameproj))
    return (project_list, project_list, desc_list, "multi")
```

You can also see how Waf uses hinting by engaging Waf in GUI mode and entering the following command: `lmbr_waf.bat show_option_dialog`

This displays an options dialog box that you can review to determine hinting.

Adding Qt 5 Content to Waf

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can add Qt 5 content into the Waf build system. Typically you use an IDE (integrated development environment) tool such as Qt Designer to create and edit the Qt source file. As with all files that are processed through the Waf build system, the Qt source file must be included in the corresponding *.waf_files file for each project.

Intermediate files that need additional compilation such as the .rcc file from the .qrc compiler do not need to be specified explicitly in these files or any other source file. In addition, intermediate .rcc files are never included in any uber files (if the uber file option is enabled) since they are not compatible with uber files in general.

To enable Qt for a particular module, you must define it as a feature. Each module's configuration is in a wscript file that is in each module's directory. For example, the EditorUI_QT module has a wscript file located at dev\Code\Sandbox\Plugins\EditorUI_QT\. To enable Qt, edit this file and add ['qt5'] to the features line, as shown in the following example.

```wscript
bld.CryEditorUiQt(
    # Settings
    target = 'EditorUI_QT',
    vs_filter = 'Sandbox/Plugins',
    file_list = 'editorui_qt.waf_files',
    platforms = ['win'],
    configurations = ['debug', 'debug_test', 'profile', 'profile_test'],
    features = ['qt5'],
)
```

**MOC (Meta-Object Compiler) Files**

When header files need to be processed by the Meta-Object Compiler (MOC) as part of the build process, the build system identifies them by including their MOC output file inside the source .cpp file. For example, if foo.h is a file that is to be processed by MOC, then the source foo.cpp file also needs to include the corresponding #include for the .moc file that is generated.

For example:

```cpp
...
#include "foo.h"
...
#include <foo.moc>
```

The #include for the .moc file requires angled brackets because the generated .moc file does not reside in the local project directory but rather is located in an intermediate directory. Also, the include path that is added to the project is based on the mirrored project base in the intermediate directory. If the header file exists in a relative subdirectory, that subdirectory needs to be included in the #include for the .moc, regardless of where the .cpp file is located.

For example, if foo.h and foo.cpp are moved into the \test subdirectory, the result looks like the following:

```cpp
...
#include "foo.h"  // This can still be relative to the current source file
...
#include <test/foo.moc>  // This needs to be relative to the base path for the project in the intermediate directory.
```
QRC (QT Resource Collection) files

Qt resource collection (.qrc) files are processed by the Qt .qrc compiler. The output file has the same source name but with an .rcc extension. The resulting .rcc file is stored in the projects intermediate directory relative to any subdirectory that it exists in.

For example, if the file foo.qrc is located in the \test subdirectory, the generated .rcc file is stored in the \test subdirectory under the project's intermediate directory structure. There is no need to explicitly include the generated .rcc file into any source file as it is added as a build task for the project.

The following is an example of the contents in an ObjectPanel.qrc file:

```xml
<RCC>
  <qresource prefix="/Panels/ObjectPanel">
    <file alias="icon_layers.png">res/icon_layers.png</file>
  </qresource>
</RCC>
```

For more information, see The Qt Resource System documentation.

UI Files

Designer UI files are processed by the Qt UIC (user interface compiler). The output file has an .h header extension to it, and ui_ is also added to the name of the source. The resulting header file is created in the project's intermediate directory relative to its location in the project.

For example, if the file foo.ui is located in a \test subfolder, the generated ui_test.h file will be located in the \test subfolder under the project's intermediate folder structure.

When including the generated header file, using the same rule as the moc include applies as follows:

```c
#include "foo.h"
...  
#include <test/ui_foo.h>  // Path is relative to the project root
```

Qt Linguist (TS) files

Qt Linguist files (.ts) are processed by Qt and output as .qm files. The .qm files are automatically included into a single .qrc file specified by the langname attribute in the wscript file. The .qrc file is automatically added as a build task like other .qrc files for the project.

The following example demonstrates adding the required langname attribute to a wscript file:

```python
...  
def build(bld):
    bld.YourPlugin(
        ...  
        langname = 'en-us',
        ...
```

The .qm files are loaded using the QTranslator module, and the Qt resource directory is the same relative to the source directory. For example, if there a foo_en-us.ts file in a \test subdirectory, then that is the same directory that you use when loading the resource, as shown in the following example:

```c
...  
#include <QTranslator>
```
Using Uber Files

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Uber files combine multiple C and CPP files into a single compilation unit, which is intended to reduce input/output impact on compilation time and help accelerate build time.

The code in uber files must meet the following coding standards:

- No global statics in the global namespace
- No global 'using namespace' declarations

Waf compile jobs include files from the *.waf_files lists. These files have the following format:

```json
{
    "<uber_file>": {
        "<source_filter_name>": [
            "file1.h",
            "file1.cpp"
        ]
    }
}
```

Valid values for `<uber_file>` are:

- **none** – Files in this list are banned from uber files. If you want your module to use precompiled headers, you must include them in this list.
- **auto** – Files in this list are combined into modules that are optimized for compile time by Waf. Files that are automatically combined are sorted by absolute path and then combined until the file size path is reached. The combination must be deterministic given the same input files and file size limit.

File size limits vary depending on the compilation:

- 200K – Suggested for compiling remotely using Incredibuild
- 300K – Default setting and suggested for compiling locally using an SSD
- 400-500K – Suggested for compiling using an HDD

You can specify the file size by updating the `uber_file_size` value in the `user_settings` file or by running the following command: `--uber-file-size <filename>.cpp`

- **somefilename.cpp** – Files in this list are combined into `somefilename.cpp`. This action is useful when certain files can only be combined together or when you want to combine platform-specific code.
Using the Waf Build System

Most waf_files lists should include one none section with the precompiled header and an auto section with everything else.

**Configuring Waf**

To help obtain the most optimal compile times, use the following:

- `use_uber_files = True`
- `max_parallel_link = 4`
- `use_incredibuild = True`
- `incredibuild_max_cores = 128`

**Debugging Waf**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you encounter issues that are not related to configuration, it is important to debug the internal Waf library. For a Python callstack, you typically must debug in `\dev\Tools\Build \waf-version\waflib`.

Using PyCharm, an IDE for Python development, you can browse to a file where you are having problems, set a breakpoint, and click the bug icon to start debugging. Execution time may be slower when running PyCharm.

Opening the root directory creates file indexing. You can use PyCharm to specify folders to exclude from the project structure, as shown in the example image.
You can also debug the way you would any native Visual Studio solution-based project. Right-click the project you want to debug and select **Set as Startup Project**. Continue the debugging process as you normally would. If you receive a warning that the _WAF_ project is outdated but your project is already up-to-date, click **No** to build.

**Troubleshooting**

When using multiple jobs (for example, `--jobs=12`), Waf can be difficult to debug. Try using `--jobs=1` to disable multi-threading.

When using IncrediBuild, the debugger won't properly execute all break points. Try disabling IncrediBuild when debugging Waf.

**Adding Custom Game Icons**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add a custom icon that appears in the top left title bar window of your game.
To add a custom game icon

1. Create an icon and name it `default_icon`. You can save it in `.tif`, `.png`, `.tga`, or `.bmp` format.
2. Save the icon file to your game's `\textures` directory.

Using Lumberyard sample projects and levels

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard provides a variety of sample projects, levels, and assets, which are located in the `\lumberyard_version\dev` directory.

Use the Project Configurator to set or change your default project. For more information, see Creating Lumberyard projects (p. 43).

Samples Project

The Samples Project includes many sample levels that demonstrate various Lumberyard features, such as the Fur Shader (p. 1661), Rin Locomotion (p. 151), UI samples (p. 2923), and more. For a complete list and descriptions of levels, see Samples Project (p. 146).

Multiplayer Sample

You can use the Multiplayer Sample game to test the Lumberyard GridMate (p. 2080) networking features.
Sample projects and levels

In-App Purchases
The In-App Purchases sample (p. 175) demonstrates how to use the In-App Purchases (IAP) gem for a Lumberyard application that runs on a mobile device.

**Cloud Gem Samples Project**

The Cloud Gem Samples Project includes sample levels that demonstrate the use of cloud gems that can add cloud-connected functionality to your game.

Use of the following levels is covered in the associated gem's topic found in the Cloud Gems (p. 2127) documentation.

- **DynamicContentTest** – Found under CloudGemTests, this level provides a test of the Dynamic Content (p. 2155) Gem, which enables you to upload new or updated game assets to the cloud, that can be automatically downloaded to game clients.
- **PlayerAccountSample** – Demonstrates the use of the Player Account (p. 2188) Gem, which provides a standalone player authentication and management solution for your game.
- **CommunicatorSample** – Demonstrates the use of the Web Communicator (p. 2204) Gem, which can send AWS cloud service events to your game's connected editors or clients. Your editors or game clients can use this information to update themselves without having to poll AWS for updates.

**Virtual Reality Project Sample**

The Virtual Reality Project sample is a template that you can use to build VR applications for any supported device. You can download this content separately.

Legacy Sample Projects

The following sample projects and assets are available as a separate download from the Lumberyard Legacy Downloads page.

- **Woodland Asset Collection** (p. 177) – Free assets for you to use to create your levels.
• **Beach City Night Asset Collection** (p. 178) – Collection of free assets that you can use to try Lumberyard or make your own games.

• **Legacy Project (GameSDK)** (p. 180) – Enables you to use GameSDK functionality.
Samples Project

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Samples Project includes two collections of sample levels and code that demonstrates how to use various Lumberyard features.
The Samples Project's level files are located in the `lumberyard_version\dev\SamplesProject\Levels` directory.

The `Samples` subdirectory contains the following levels:

- Advanced Rin Locomotion (p. 151)
- Audio Sample (p. 147)
- Fur Technical Sample (p. 1661)
- Metastream Sample (p. 1164)
- Script Canvas Sample (p. 149)
- Scripted Entity Tweener Sample (p. 3011)
- Simple Jack Locomotion (p. 153)

The `UI` subdirectory contains the following levels:

- UI Features – Demonstrates UI components (p. 2923), UI animation (p. 3023), font rendering (p. 2999), language support and localization (p. 2415), and C++ canvases and custom components.
- UI in 3D World (p. 2897) – Demonstrates placing game UI on 3D objects.
- UI Main Menu Lua Sample (p. 2897) – Demonstrates loading a UI canvas using Lua.
- UI Main Menu Script Canvas Sample – Demonstrates loading a UI canvas using the Script Canvas editor (p. 2420).

**Audio Sample**

This sample demonstrates how to use the Audio Trigger (p. 559) and Audio Rtpc (p. 557) components with Lua scripts to associate sounds of a door opening and closing.

This example is fully annotated within the Lua script of the level file. The following sounds are associated:

- Sounds
  - Door open
  - Door creak
  - Door creak stop
  - Door slam
- Rtpc
  - Creak volume
  - Creak pitch

To play the example, do the following:

- Press W/S to swing the door.
- Press Q to open the door.
- Press E to open the door.
- Press the `spacebar` to open or close the door.

To see the Lua script, select the Door entity (a child of DoorTest) and then click the `{ }` button next to the script property to open the Lua IDE.
For more information about audio, setting up sounds and using Wwise LTX, see Adding Audio and Sound Effects (p. 3140).

**Starter Game Sample**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Starter Game sample to see how Lumberyard systems work together to make a game. Starter Game is a small, third-person game that is built with the Lumberyard component entity system. In addition to component entities, Starter Game demonstrates bipedal locomotion, the Dynamic Vegetation system, the Script Canvas visual scripting system, use of PhysX as the physics system, and more. In this sample, you play as Valena, a human who has crashed on a distant planet. Valena can explore the world and must defend herself against enemy robots that occupy the planet.

Starter Game is included in the Lumberyard installation package. It runs on PC and supported consoles, but is not supported for MacOS and mobile devices.

All reusable assets and scripts have been placed into the following Asset Gems, so you can easily add them to your own projects. You'll find these under `dev\Gems\StarterGame`:

- Starter Game Audio
- Starter Game Characters (includes animations)
- Starter Game Environment
- Starter Game User Interface
- Starter Game Weapons

**To run the Starter Game sample**

1. Set Starter Game as the default project:
a. Double-click the Project Configurator desktop icon.
b. In the Project Configurator, select **Starter Game**.
c. Choose **Set as default** and close the Project Configurator.

2. Run Lumberyard Editor.

3. In the **Welcome to Lumberyard Editor** dialog box, choose **Open level**.

4. In the **Open a Level** dialog box, open the **Game** folder, select the **SinglePlayer** level, and choose **Open**.

5. To start the game, press **Ctrl+G**.

6. Use the following keyboard keys and mouse controls:

<table>
<thead>
<tr>
<th>Input</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>W, A, S, D</td>
<td>Move Valena either parallel or orthogonal to camera.</td>
</tr>
<tr>
<td>Shift</td>
<td>Hold this key to walk instead of run.</td>
</tr>
<tr>
<td>Spacebar</td>
<td>Jump.</td>
</tr>
<tr>
<td>Mouse</td>
<td>Orbit camera.</td>
</tr>
<tr>
<td>Primary mouse button</td>
<td>Fire weapon.</td>
</tr>
<tr>
<td>Secondary mouse button</td>
<td>Aim weapon.</td>
</tr>
<tr>
<td>Esc</td>
<td>Exit the game.</td>
</tr>
</tbody>
</table>

If you're running Starter Game in the editor and Valena dies, press **Esc** to exit the game.

**Note**
To toggle debugging information on and off in the viewport when you're in gameplay mode, press the tilde (~) key. For more information, see **Using Console Debug Views** (p. 1967).

## Script Canvas Basic Sample

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Script Canvas Basic Sample contains a collection of five Script Canvas scripts that demonstrate the following gameplay functions:

- Basic player controller
- Ballistic projectile
- Door that opens based on a trigger volume
- Light that toggles based on a trigger volume
- Flickering light
To see the Script Canvas graphs

1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the Script Canvas editor, choose File, Open.
3. Navigate to the `lumberyard_version\dev\SamplesProject\Levels\Samples\ScriptCanvas_Sample\ScriptCanvas_Basic_Sample\ScriptCanvas` directory.
4. Open one of the Script Canvas scripts:
   - doortriggerarea.scriptcanvas
   - flickerlight.scriptcanvas
   - lumbertankcontrols.scriptcanvas
   - projectile.scriptcanvas
   - tooglelight.scriptcanvas

To play the game

Do the following:

- To start the level, press Ctrl+G.
- To control the tank, do the following:
  - To move forward and backward, press the W and S keys, respectively.
  - To turn left and right, press the A and D keys, respectively.
  - To control the camera, move the mouse pointer.
  - To fire the tank's gun, use the left mouse button.
- To open the door, drive up to it or shoot it.
- To turn off the light, drive up to the light.
- To turn the light on, drive away from the light.
- To exit the level, press Esc.
Advanced_RinLocomotion Sample

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This sample level showcases the Rin character in a simplified environment and demonstrates how the Animation Editor uses actors, motion sets, and animation graphs to control character behavior. Playable animations include idle, walk, run, turn, and multiple attacks.

The Advanced_RinLocomotion sample level is located in the \dev\SamplesProject\Samples \Advanced_RinLocomotion directory. The assets are located in the \dev\SamplesProject \AnimationSamples\Advanced_RinLocomotion directory.

Once the sample level is open, in the Perspective viewport in Lumberyard Editor, use the following keyboard keys and mouse controls:

- To start the level, press Ctrl+6.
- To move forward, left, backward, and right, press the W, A, S, and D keys, respectively.
- To look around, move the pointer.
- To attack, press the left mouse button.
- To jump, press the Space key.
• To exit game mode, press Esc.

You can also use a controller connected to your computer:
• To control the movement direction, use the left joystick.
• To control the camera navigation around the character, use the right joystick.
• To attack, press X.
• To jump or dive roll, press A.
• To zoom in the camera, press the right trigger.
• To zoom out the camera, press the left trigger.
• To snap the camera to the back of the character, press the right joystick.
Simple_JackLocomotion Sample

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This sample level showcases Jack the robot in a basic environment and demonstrates how the Animation Editor uses actors, motion sets, and animation graphs to control character behavior. Playable animations include idle, walk, run, and navigate.

The Simple_JackLocomotion sample level is located in the \dev\SamplesProject\AnimationSamples directory. The assets are located in the \dev\SamplesProject\AnimationSamples\Simple_JackLocomotion directory.

Once the sample level is open, in the Perspective viewport in Lumberyard Editor, use the following keyboard keys and mouse controls:

- To start the level, press Ctrl+G.
- To move forward, left, backward, and right, press the W, A, S, and D keys, respectively.
- To look around, move the pointer.
- To toggle the LookAt node on and off during the idle animation, press Tab.
- To aim, hold the left mouse button.
- To exit game mode, press Esc.

You can also use a controller connected to your computer:

- To control the movement direction, use the left joystick.
- To control the camera navigation around the character, use the right joystick.
- To toggle the character aim on and off, press the right trigger.
- To toggle the LookAt node on and off during the idle animation, press the left trigger.
Multiplayer Sample

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Multiplayer sample shows you how to build and structure multiplayer games that use the various features of the GridMate networking (p. 1990) library. The new preview attempts to incorporate as many GridMate features as possible but is not comprehensive. Instead it is intended as a living snapshot of the current state of the networking library. Consequently, as the sample evolves, features might be added to enhance functionality or removed to keep the sample concise.

Building and Running the Sample

Follow these steps to build and run the sample.

To build and run the Multiplayer sample

1. In the Project Configurator set MultiplayerSample as the active project.
2. Run the following command:
   ```
   lmbr_waf configure
   ```
3. Run the following command which builds everything in the profile, for your version of Visual Studio.
   ```
   lmbr_waf build_win_x64_vs2017_profile -p all
   ```
4. Run the MultiplayerSampleLauncher.exe file from your default bin directory.
   - For Visual Studio 2017, use the Bin64vc141.Dedicated directory.
   - For Visual Studio 2019, use the Bin64vc142.Dedicated directory.

MultiplayerLobby

This level demonstrates a multiplayer lobby that uses GridMate networking. Currently, the level performs the following tasks:

- Creates a server
- Displays a list of servers on the local LAN
- Connects to a server

To create a server

1. Enter the server name in the Create a Server form.
2. Click Create Server.

   The game starts hosting and loads the selected map.

To connect to a server

1. Find your server in the server browser list. If your server doesn't appear, click Refresh.
2. To select your server, click the row that contains your server name.
3. Click **Connect**.

**Game Overview**

The Multiplayer sample is an arcade space shooter. Each player controls a ship in a giant field of asteroids. Each player must fly through the asteroids and destroy as many of them as possible in an effort to achieve the highest score. Additional points are awarded when other players are killed. Points are deducted every time the player dies. In addition to the threats posed by asteroids and other players, gravity wells scattered around the map can draw players in and trap them. Players who become trapped in a gravity well die instantly.

When a ship is spawned, three kinds of ships are assigned randomly:

- **Fighter** – An average space ship.
- **Speeder** – A faster and more maneuverable space ship but less sturdy as a result.
- **Rammer** – A sturdier ship that is weaponless but designed to smash into asteroids and destroy them. The ship is fast but has poor maneuverability.

When destroyed, some asteroids spawn collectibles that award a short-time power-up bonus. These power-ups include extra damage against certain targets, a longer power-up time, and extra protection against certain sources of damage.

**Game Controls**

The following tables list controls for keyboards, controllers, and touch interfaces.

**Keyboard**

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASD keys</td>
<td>Move the ship relative to the screen (W is always up, S is always down, and so forth).</td>
</tr>
<tr>
<td>Arrow left, arrow right</td>
<td>Rotate the ship clockwise or counterclockwise.</td>
</tr>
<tr>
<td>Spacebar</td>
<td>Fire the ship's weapon (if applicable).</td>
</tr>
<tr>
<td>Tab (hold)</td>
<td>Open the <strong>Player Stats</strong> window.</td>
</tr>
</tbody>
</table>

**Controller**

<table>
<thead>
<tr>
<th>Control</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left joystick</td>
<td>Move the ship in absolute terms relative to the screen.</td>
</tr>
<tr>
<td>Right joystick</td>
<td>Rotate the ship clockwise or counterclockwise.</td>
</tr>
<tr>
<td>Right shoulder button</td>
<td>Fire the ship's weapon (if applicable).</td>
</tr>
<tr>
<td>Back button (hold)</td>
<td>Open the <strong>Player Stats</strong> window.</td>
</tr>
</tbody>
</table>
Touch interface

<table>
<thead>
<tr>
<th>Control</th>
<th>Action</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual joystick (left side of screen, no visual indication)</td>
<td>Move the ship in absolute terms relative to the screen.</td>
<td></td>
</tr>
<tr>
<td>Virtual joystick (right side of screen, no visual indication)</td>
<td>Rotate the ship clockwise or counterclockwise.</td>
<td></td>
</tr>
</tbody>
</table>

Notes

• As soon as you begin moving with either virtual joystick, the ship begins firing (if applicable).
• These controls can also be accessed by using the mouse (click to touch).
• If you fire manually, the automatic firing stops until the next input from one of the virtual joysticks.

Game CVars

The following console variables can be used with the game.

mps_AISteeringDebug

Specifies whether the debug drawing information for the bot is drawn.

mps_ControllerType number

Controls the type of PlayerController that the client uses to control the assigned ship. The following table shows the possible values for number:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Specifies manual control. A user controls a single ship.</td>
</tr>
<tr>
<td>1</td>
<td>A bot that plays the game.</td>
</tr>
</tbody>
</table>

mps_DrawBucketSystem

Specifies whether the debug information for the bucket system is drawn.

mps_SpawnTag

Controls what ship is spawned for a player when the player spawns a ship. Possible values are shown in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>Chooses an available ship at random. This is the default value.</td>
</tr>
<tr>
<td>Fighter</td>
<td>Spawns a fighter ship.</td>
</tr>
<tr>
<td>Speeder</td>
<td>Spawns a speeder ship.</td>
</tr>
<tr>
<td>Rammer</td>
<td>Spawns a rammer ship.</td>
</tr>
</tbody>
</table>
mps_User\textit{Name}

Specifies the name for the client in game elements such as the scoreboard or action notification.

**Creating a Dedicated Server**

To create a dedicated server for the game, perform the following steps.

**To create a dedicated server**

1. In the Project Configurator set \textit{MultiplayerSample} as the active project.
2. Enter the following command:
   
   \texttt{lmbr\_waf configure}

3. Enter the following command which builds everything in the profile, for your version of Visual Studio:
   
   \texttt{lmbr\_waf build\_win\_x64\_vs2017\_profile -p all}

4. Enter the following command which builds \texttt{game\_and\_engine} in \texttt{profile\_dedicated}, for your version of Visual Studio:
   
   \texttt{lmbr\_waf build\_win\_x64\_vs2017\_profile\_dedicated -p game\_and\_engine}

5. Run the \texttt{MultiplayerSampleLauncher\_Server.exe} file from your default bin directory.
   
   • For Visual Studio 2017, use the \texttt{Bin64vc141.Dedicated} directory.
   • For Visual Studio 2019, use the \texttt{Bin64vc142.Dedicated} directory.

**Supported Platforms**

The Multiplayer sample is currently supported on a variety of client and server platforms.

**Clients**

The following playable client platforms are supported:

• Windows (Visual Studio 2017 or Visual Studio 2019)
• Android
• iOS

**Dedicated Server Platforms**

The following dedicated server platforms are supported:

• Windows (Visual Studio 2017 or Visual Studio 2019)
• Linux (Ubuntu)

**About Self-Signed Certificates in the Multiplayer Sample**

In Windows, the Multiplayer sample enables client-side self authentication that uses a \texttt{self-signed certificate}, which the build script generates for each user. Regarding this certificate, keep in mind the following points:
• The self-signed certificate is not an official certificate. If you want to ship a publically available game, you must replace it with a public key certificate.

• On the macOS and Linux platforms, you must generate self-signed certificates manually.

• On the macOS and Linux platforms, the Multiplayer sample has the following limitations:
  • macOS: Only iOS client builds can be made.
  • Linux: Only dedicated server builds can be made.

Troubleshooting

The following are some connection troubleshooting tips.

• If clients are unable to discover a server, ensure that the server and client are on the same subnet in order to allow for UDP discovery.

• If clients are unable to connect to a server, ensure that the server and client are using the same key and certificate files.

• When trying to host or join a session, you might receive one of the following error messages:

  Invalid Secure Socket configuration given for hosting a session.
  Ensure that a Public and Private key are being supplied.

  Invalid Secure Socket configuration given for joining an encrypted session.
  Ensure that a Certificate Authority is being supplied.

To resolve these issues, perform one of the following tasks:

• Run the command `lmbr_waf configure` to generate a set of certificates.

• In the `MultiplayerSample/certificates` directory, provide your own appropriately named certificates.

For more information about self-signed certificates, see Encryption in the Multiplayer Sample Network Features (p. 162) section and About Self-Signed Certificates in the Multiplayer Sample (p. 158).

Topics

• Multiplayer Sample Gameplay Architecture (p. 159)
• Multiplayer Sample Network Features (p. 162)

Multiplayer Sample Gameplay Architecture

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Multiplayer sample consists of five main entities and an equal number of supporting gameplay systems. The following overview provides a conceptual introduction to the architecture of the Multiplayer sample.

Principal Entities in the Multiplayer Sample

The principal entities in the Multiplayer sample are ships, asteroids, gravity wells, bullets, and collectibles. These entities also have some common components.
Common Components

The principal entities in the game share the following components:

- **SimpleSpacePhysicsComponent** – This component enables objects to move around in space and be affected by gravity.
- **CollisionSystemComponent** – These components enable objects to interact with each other in the world.
- **NetBindingComponent** – Activates the net binding code of other components on an entity and signals the game that the entity should be replicated across the network.

Ships

The Multiplayer sample uses a variety of ships that demonstrate how a selectable game state can be synchronized from a client to a session when the player first joins the game. Different types of ships also provide a variety of game experiences. Some ships have long range weapons, and others are short range and have no weapons. The following components are related to ships:

- **ShipComponent** – Identifies an object as a ship to Lumberyard. It also takes commands from the player and converts them on the host into the motion required to move the ship.
- **ShipGunComponent** – Handles the firing of bullets.
- **ActiveModifiersComponent** – Manages the modifiers that affect the ship.
- **CollectibleInterpreterComponent** – Enables a layer of interpretation between a collectible and the actual modifier that is granted. This enables implementations of a collectible according to the ship type. For example, ships with guns that receive a collectible get a special laser, while weaponless ships get a nonweapons-related buff.
- **HealthComponent** – Determines the relative health of the ship as long as it remains operational.

Asteroids

Asteroids are the most common slice in the Multiplayer sample. They exist in three sizes (small, medium, and large). These asteroid types are functionally the same while they exist. When destroyed, they react differently by spawning varying amounts of asteroids and collectibles. Blanketing the playfield with these objects allows for optimization techniques to be implemented that create a reasonable visual experience without overusing bandwidth. The following components are related to asteroids:

- **AsteroidComponent** – Identifies an object as an asteroid to the game engine.
- **DeathActionComponent** – Allows for actions to be taken upon the destruction of the object. In the sample, the DeathActionComponent spawns additional asteroids and collectibles.
- **DamageComponent** – Determines how much damage the asteroid does when it collides with a ship.
- **HealthComponent** – Determines how much damage an asteroid can take before it is destroyed.

Gravity Wells

Gravity wells create action inside the game by causing objects to be destroyed and spawned even in the absence of user interaction. This prevents the game state from falling into stasis. The following components are related to gravity wells:

- **GravityWellComponent** – Identifies an object as a gravity well.
- **GravitySourceComponent** – Applies gravity to physical objects and draws them towards the source object.
Bullets

Ships use bullets to damage asteroids and each other. The use of bullets drives gameplay and generates points. The following components are related to bullets.

- BulletComponent – Identifies an object as a bullet.
- DamageComponent – Determines how much damage a bullet does when it hits a ship or asteroid.

Collectibles

Collectibles add fun and excitement to the game and provide a way to enable and test dynamic in-game changes to otherwise static systems. The sample uses collectibles to test the handling of constantly shifting objects to find the right balance between natural appearance and controllability. The following component is related to collectibles.

- CollectibleComponent – Identifies an object as a collectible.

Multiplayer Sample Gameplay Systems

The gameplay systems in the Multiplayer sample handle the game logic essential to game operations.

Game Manager

The GameManagerComponent handles all of the general game logic, such as controlling game start and end, managing the results screens, and other tasks not performed by individual game modes. The GameManagerComponent functions as a server-only component. A component like this would normally be reserved for use in a dedicated server and not be shipped in a player version of the game. The component synchronizes no data and is not replicated across the network.

Score Attack Game Mode

The ScoreAttackGameModeComponent works with the GameManagerComponent to provide the rules for play and synchronize data to players for score attack mode.

Asteroid Manager

The AsteroidManagerComponent creates new asteroids and manages asteroid lifespans. It decides where, how, and how many asteroids are spawned into the game. Although it is exported with the level, this component synchronizes no data and should not be included in a player version of the game.

Collectible Manager

The CollectibleManagerComponent spawns collectibles in response to the destruction of asteroids. Although it is exported with the level, this component synchronizes no data and should not be included in a player version of the game.

Spawn Manager

The SpawnManagerComponent is a wrapper for the Spawner (p. 859) that allows the sample to spawn objects in complex ways without requiring the use of multiple spawner components. The spawn manager facilitates the automation of variation among spawned entities. For example, when an asteroid is spawned, a general type can be specified and the specific type randomly selected in the spawn manager from the supported types. This removes the need to manually specify a type with each spawn request.

This approach also promotes ease of maintenance because it enables a single object to handle all objects of a single type. For example, all asteroids are defined in a single place and are identified by a specific tag (SmallAsteroid, MediumAsteroid, or LargeAsteroid). This limits the number of interactions required among the objects and the amount of information that must be exchanged.
In the sample, only one object manager (for example, asteroid or collectible) is aware of the location of the spawner for an object. This location is usually on the object itself.

**Multiplayer Sample Network Architecture**

The sample uses a server-authoritative architecture. In the sample, client-side input is processed on the player and converted into RPCs that are processed by the server.

**Player Object**

When each client joins the session, it creates an object on the server that represents the player. The object provides information relevant to player configuration. The client maintains full control over player configuration, which the server reads from the object. The player object consists of a `GamePlayerComponent` and a related `GamePlayerChunk`.

- **GamePlayerComponent** – The `GamePlayerComponent` represents the player on the server. The component is owned by the client and replicated to the server. This component handles local player configuration such as the player's name and the ship to spawn. The user can change this information during gameplay. These changes must be reflected to the server.

- **GamePlayerChunk** – The Replica Chunks (p. 2037) that represents player information on the server. The `GamePlayerChunk` specifies the display name of the player and the ship that the player wants to spawn.

**Ship Object**

The ship object consists of a `ShipComponent` and a related `ShipComponentReplicaChunk`.

- **ShipComponent** – Manages the overall ship logic, controls, and ship configurations.

- **ShipComponentReplicaChunk** – Contains the RPCs that are invoked on the client and passed to the server. The server then validates, sanitizes, and applies the results to the game state.

**See Also**

For more information about networking in the Multiplayer sample, see Multiplayer Sample Network Features (p. 162).

**Multiplayer Sample Network Features**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Multiplayer sample offers the following features.

- **Dedicated Server Split (Partial)** – Because the clients can also host game sessions for LAN play, a true split is never done even when the client has no server code. However, the code itself is compartmentalized in such a way that the split can be done easily. A dedicated server split is used in the following component:

  - **GameManagerComponent** – Designed to be available only on the server. It relies on the GameModeBus to synchronize the state to display (for example, whether the game is ready to start a round, or whether the current round is complete). For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic` directory.

- **Client Authoritative Control** – Enables the client to maintain exclusive control of an object whose actions are processed on the server. Client authoritative control is used in the following components:
• **ShipComponent** – Client commands that control a ship trigger a series of RPCs that the ShipComponent exposes. The RPCs are then processed on the server. The commands are restricted to the owner of the object. For details, see the source code files in the `dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\Ship` directory.

• **ManualPlayerControllerComponent** – Represents the human-controllable player. For details, see the source code files in the `dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\ShipController` directory.

• **BaseAutomatedPlayerControllerComponent** – Represents an AI-controllable player. It maps computer-generated input to the RPC calls. For details, see the source code files in the `dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\ShipController` directory.

• **Encryption** – Encryption is performed using `FileDataSource` with self-signed certificates and certificate pinning. The files expected are a set of X.509 PEM files, which are split into a nonpassword-protected key (`multiplayersample.key.pem`) and a cert file (`multiplayer.cert.pem`). These are located in the `MultiplayerSample\certificates` directory. If either of these files is missing, a set of certificates is generated using RSA2048 and some default answers that can be customized from the `wscript` file. For more information, see How To Generate a Private Key and Public Certificate (p. 2072). For important information about self-signed certificates in the Multiplayer sample, see About Self-Signed Certificates in the Multiplayer Sample (p. 158). In the Multiplayer sample, encryption is used in the following features:

• **GameManager** – Exposes a function called `SetupEncryption`, which handles the configuration of certificates for the game. For details, see the source code files in the `dev\Code\MultiplayerSample\Game\Game` directory.

• **MultiplayerUtils** (in Multiplayer Gem) – Has utility functions that handle configuration of the carrier description (p. 2014) for hosting and joining. The code shows how to configure the `SecureSocketDriver` (p. 2070) and use an EBus event to signal its use for the connection. For details, see the source code files in the `dev\Gems\Multiplayer\Code\Include\Multiplayer` directory.

• **RPC Traits** – In Lumberyard, RPCs allow games to send events or requests to remote nodes through GridMate replicas (p. 2034) that synchronize the state of the session. In the Multiplayer sample, RPC traits are used to control ships, manage the HUD state, and manage audio controllers. To see how they are attached to RPCs, see the source code files in the following locations.

<table>
<thead>
<tr>
<th>Component</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShipComponent</td>
<td><code>dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\Ship</code></td>
</tr>
<tr>
<td>CoreGameHUDControllerComponent</td>
<td><code>dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic</code></td>
</tr>
<tr>
<td>ShipAudioControllerComponent</td>
<td><code>dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\Audio</code></td>
</tr>
</tbody>
</table>

• **Custom Component Net Binding** – For a Lumberyard component to share data on the network, it must include the `NetBindingComponent` (p. 2055). A number of Multiplayer sample components demonstrate custom component net binding.

• **CoreGameHUDController** – An illustrative example that has datasets, RPCs, RPC traits, and callbacks. For details, see the source code files in the `dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic` directory.

• **Custom Type Marshalling** – In GridMate, all data marshalling (p. 2022), whether for a dataset or an RPC, is written using a specialized `Marshaler` type. If the type is a complex type like a class or container, then that marshaler marshals each of its fields with nested marshalers. Custom marshalers...
can be implemented to support custom types. Custom type marshalling is used in the following component:

- **ScoreAttackGameModeComponent** – The `PlayerRoundInfo` struct demonstrates the simplest way of defining a custom marshaler for a custom type without needing to specifically override it in the dataset definition. For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic` directory.

## Virtual Reality Samples Project

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Virtual Reality Samples Project is a template that you can use to build virtual reality (VR) applications for any supported device. The sample is configured with the base set of gems (HMD Framework, Oculus, and OpenVR) that you need for VR and demonstrates the following essential VR features:

- Room-scale VR setup
- Tracked controller input system
- Spatialized audio playback
- Stereo 360 video playback

To install the Virtual Reality Samples Project

1. Download the `vrproject.zip` package at Lumberyard Downloads and extract it in your Lumberyard directory, such as `lumberyard_version\`
2. Open Lumberyard Setup Assistant (p. 16) and on the Summary page, click Configure project.
Note
To ensure that the VR project launches, you must use Lumberyard Setup Assistant to open the Project Configurator. Lumberyard Setup Assistant copies required SDKs from the 3rdParty directory into the dev\VirtualRealityProject directory.

3. In the Project Configurator, select VirtualRealityProject and then click Set as default.
5. Open Lumberyard Setup Assistant.
6. Download the required SDKs to view video playback for the Virtual Reality Samples Project.

Important
The video playback on the TV in the VR_TVRoom_Sample level must be enabled before you can view video playback. For instructions on setting up video playback, see Setting Up Video Playback (p. 907).

7. Close Lumberyard Setup Assistant.
8. After completing video playback setup, you must also build (p. 61) the virtual reality project.
9. Start Editor.exe from one of the following directories:
   - For Visual Studio 2017, use the lumberyard_version\dev\Bin64vc141 directory.
   - For Visual Studio 2019, use the lumberyard_version\dev\Bin64vc142 directory.

Topics
- VR Islands Level (p. 165)
- VR TV Room Level (p. 167)
- VR Xylophone Level (p. 169)
- VR Box Garden Level (p. 172)

VR Islands Level

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The VR Islands level (VR_Islands_Sample) shows you how to create a simple VR level with the instantvr entity and physics collision on controllers. This level features floating islands between which a player can teleport with the trigger button on the Oculus or Vive motion controllers. The player can hold the trigger button to pick the location and then release the button to initiate the teleport.

Note
This level uses Script Canvas and Lua for game logic. Legacy systems have been replaced or removed, with the exception of the navigation mesh. The VR Preview (p. 924) component generates the navigation mesh, which is required for teleporting in gameplay.

This level uses an instantvr slice that has the following child entities:

- Controller_One – Represents the player's right hand.
- Controller_Zero – Represents the player's left hand.
- Camera – Represents the player's head.
- InvalidLocationSpawner – Spawns an invalid location marker when the player teleports within the terrain or navigation mesh.
• **ValidLocationSpawner** – Spawns a valid location marker when the player teleports within the terrain or navigation mesh.
• **InputConfiguration** – Contains the input bindings for teleportation events.

For more information about installing the project that includes the VR Islands level, see Virtual Reality Samples Project (p. 164).

**Using the InstantVR Entity**

The VR Islands level is created with the `instantvr` entity, which has the following components:

- **Lua Script (p. 683)** – This component uses the `instantvr.lua` script for VR device tracking and teleportation functionality. You can find the script in the `lumberyard_version\dev\VirtualRealityProject\Scripts\vr` directory.

- **VR Preview (p. 924)** – This component automatically generates a 50x50 meter navigation area, and a navigation mesh within the navigation area. The navigation mesh is required for the `instantvr.lua` script to detect valid teleportation locations.
Note
You must select the Static check box on the Transform (p. 878) component for all entities in the navigation area. This ensures that the navigation mesh generates correctly. The Static check box is cleared by default.

Using the HideOcean Entity

The VR Islands level includes a HideOcean entity that allows you to hide the ocean when the level runs in VR mode.

To hide the ocean

1. Open the Console Variables window. For instructions, see Viewing the Console Window (p. 210).
2. In the Console Variables window, search for the e_WaterOcean console variable.
3. Set the e_WaterOcean console variable to 0.

VR TV Room Level

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The VR TV Room level demonstrates a simple VR level that is set up to play 2D video, 3D video, and 360 video on a VR device. This level contains the following:

- 2D video
- 3D video
- Input Configurator entity – Initiates an input event when you press the trigger button.
- VideoPlayback_Toggle entity – Contains a Script Canvas (p. 818) component that listens for the input event and then toggles visibility.
- Screen_Toggle_Sphere entity – Contains a Lua script that enables the 3D video sphere to follow the VR camera.

For more information about installing the project that includes the VR TV Room level, see Virtual Reality Samples Project (p. 164).

Using Components in the VR TV Room Level

You can use the Video Playback (p. 906) component to assign a diffuse texture to the specified video. You can then assign the material that uses the diffuse texture to any mesh.

You can use the Script Canvas component and graph to set the playback to loop mode and to start playing immediately.
Cycling Through Video Playback Options

The TV Room level shows you how to cycle through video playback options. When you use the trigger button in-game, the television cycles through 2D video, 3D video, 360 video, and an off state. In Lumberyard, this is achieved with four entities, one for each video playback option. You can toggle the visibility of each entity to reflect the playback on the television in the level.

The 2D and 3D videos were created with the same static mesh, but different materials. The 360 video was created by attaching a sphere to a camera with a 3D video material.

VR Xylophone Level

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The VR Xylophone level shows you how to create basic VR gameplay with the Lumberyard component entity system. You use Script Canvas and Lua for VR device tracking, input events, and corresponding gameplay events.

For more information about installing the project that includes the VR Xylophone level, see Virtual Reality Samples Project (p. 164).
VR Device Tracking

The VR Xylophone level uses the following:

- **Input Configurator** entity – Handles VR device tracking and input events.
- **Input (p. 642)** component – Defines two input events (**FireLargeBox** and **FireSmallBox**) that are initiated when you use the trigger button on the Oculus or Vive motion controllers.
- **Lua Script (p. 683)** component – Uses the `vrdevice_tracking.lua` script to track the movement and orientation of the VR controllers. Passes the transforms to specific controller entities.
**VR Controllers and Projectiles**

The VR Xylophone level uses a controller_right.slice for the right controller and a controller_left.slice for the left controller. You can find these slices in the lumberyard_version\dev\VirtualRealityProject\slices directory.

The right and left controllers are set up similarly, except that they spawn different projectiles and use different materials. The following image shows the entities for the left controller (green) and right controller (red), as well as their respective projectile entities.

![VR Controllers and Projectiles](image)

The projectile entities are also set up similarly, except for size and color. They use the Rigid Body component, Primitive Collider and Box Shape (p. 820) components to collide and interact with the domino entities in a physically realistic way. Upon spawning, the projectile entities use the Script Canvas (p. 818) component to move forward.

You can find the projectile_largebox.slice and projectile_smallBox.slice files in the lumberyard_version\dev\VirtualRealityProject\slices directory.

The VR Xylophone level uses the following scripts:

- **vrdevice_tracking Lua script** – Matches the in-game movement and orientation of the controller entities with the physical controllers.
- **Script Canvas graph** – Spawns a projectile and plays a spawning sound when an input event is triggered. This graph is on the controller entity.
  - In the **Input Handler** node, the **Event Name** is the event that is defined in the **Input Configurator**.
  - The **Audio Trigger** component specifies the sound to play.
• The **Spawner** component specifies the dynamic slice to spawn.

![Diagram showing Spawner component](image)

**Domino Tiles**

The projectile entities interact with the domino tiles. These tiles are instances of eight slices, each differing in color, letter, and sound pitch. The sound pitch occurs when the tile's trigger makes contact with another collider.

Each domino tile slice has a child entity with the **Trigger Area** and **Lua Script** components. The **Trigger Area** component triggers an event to play the sound that's specified in the `triggerarea_play_sound.lua` script.

![Domino tiles image](image)

**VR Box Garden Level**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The VR Box Garden level shows you how to use the Lumberyard component entity system to set up a VR playspace, basic VR gameplay, and controller tracking. You use Script Canvas and Lua for VR device tracking, input events, and corresponding gameplay events.
For more information about installing the project that includes the VR Box Garden level, see Virtual Reality Samples Project (p. 164).

**VR Device Tracking and Controller Gameplay**

The VR Box Garden level uses the following:

- **Input Configurator** entity – Handles VR device tracking and input events.
- **Input (p. 642)** component – Defines input events that are initiated when you use the trigger button on the Oculus or Vive motion controllers.
- **Lua Script (p. 683)** component – Uses the vrdevice_tracking.lua script to track the movement and orientation of the VR controllers. Passes the transforms to specific controller entities.

The level uses a `controller_right.slice` for the right controller and a `controller_left.slice` for the left controller. These controllers listen for input events and spawn projectiles when an event is detected. The projectile entities use the Script Canvas (p. 818) component to move forward.

All slices for the VR Box Garden level are located in the `lumberyard_version\dev\VirtualRealityProject\slices` directory.

The VR Box Garden level uses the following scripts:

- **vrdevice_tracking** Lua script – Matches the in-game movement and orientation of the controller entities with the physical controllers.
- **input_play_sound** Lua script – Plays a spawning sound when an input event is triggered. The Lua Script component specifies the sound to play.
  
  **Note**
  
  To play sound correctly, you must add both the Audio Proxy (p. 557) component and the Audio Trigger (p. 559) component on the entity.

- **Script Canvas graph** – Spawns a projectile. The Spawner (p. 859) component specifies the dynamic slice to spawn. This graph is on the controller entity. In the Input Handler node, the Event Name is the event that is defined in the Input Configurator.

**VR Playspace**

The level's playspace includes the following:

- **Base** – Uses the Playspace_Base entity that has a vrplayspace_base.lua script. The Lua script adjusts the scale of the static mesh to match the dimension of the VR playspace. The dimension is derived from the position of four corners of the playspace from the VR device.
- **Posts** – Spawns at runtime from the Playspace_Posts entity. This entity uses a Lua script to calculate the number and position of posts, and spawn the posts along the playspace edges at a fixed interval.
- **Corners** – Uses a Lua script to locate the corner position and move to the correct position. There are four entities, one for each corner. You can use the Lua script to assign a number for each corner.
Procedurally-Spawned Stacking Boxes

The level uses six Box Spawner entities with the random_spawner.lua script to demonstrate how to procedurally generate a specified number of boxes at runtime. These boxes are used in-game as shooting targets.

You can modify the following parameters in the Spawner component to affect random generation.

For more information, see Spawner Component Properties (p. 859).
To enable triggering spawn events, you can add a Lua Script component to the entity with the Spawner component attached. The spawned box uses a dynamic slice called greybox.slice.

In-App Purchases Sample

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The In-App Purchases sample demonstrates how to use the In-App Purchases (IAP) Gem for a Lumberyard application that runs on a mobile device. For more information about this gem, see In-App Purchases Gem (p. 1147).

Topics

- Prerequisites (p. 175)
- Testing the In-App Purchases Sample on Android (p. 175)
- Testing the In-App Purchases Sample on iOS (p. 176)

Prerequisites

To use the In-App Purchases sample, you must have the following:

- An understanding of how to build a Lumberyard application and deploy to a mobile device. For more information, see Developing for Android and iOS with Lumberyard (p. 3166).
- Access to iTunes Connect to properly configure the sample for an iOS device.
- Access to the Google Play Developer Console to properly configure the sample for an Android device.

To load the In-App Purchases sample, you must edit the autoexec.cfg file (located in the \dev\SamplesProject directory) to include the following:

```
map Gems_InAppPurchases_Sample
```

You must also update the product IDs in the product_ids.json file to match the product IDs in iTunes Connect or the Google Play Developer Console. This file is located in the \dev\SamplesProject\IAP_ProductIds directory.

Testing the In-App Purchases Sample on Android

You can test the In-App Purchase samples on an Android device.
To test the In-App Purchases sample on an Android device

1. In a command line window, build the SamplesProject application.
2. Go to the Google Play Developer Console.
3. In the left pane, click **All Applications**.
4. In the right pane, click **Create Application**.
5. Follow the steps to create a new application. The package name for your application must match the package name under **android_settings** in the **project.json** file. This file is located in the \dev\SamplesProject directory.
6. Once the application is created, it appears on the **All Applications** tab in the Google Play Developer Console. Click the application. In the left pane, navigate to **In-app Products**.
7. On the **In-app Products** page, click **Add new product** to add products that can be purchased through the application.
8. In the left pane of the application page, click **APK** to upload the APK that you built in step 1.
9. Create a closed beta test and add test accounts for testing.
10. Deploy the application to your device. The in-app products that you created in the Google Play Developer Console appear when you select **Available Products** in the application.

You can now buy products, view purchased products, and consume purchased products in the application.

Testing the In-App Purchases Sample on iOS

You can test the In-App Purchases sample on an iOS device.

To test the In-App Purchases sample on an iOS device

1. Go to **iTunes Connect**.
   
   You must have an admin account to create applications.
2. Click **My Apps**.
3. In the top left of the page, click + to create a new application.
4. On the **My Apps** page, click your newly created application.
5. In the top pane of the application page, click **Features**.
6. In the left pane of the **Features** page, click **In-App Purchases**.
7. On the **In-App Purchases** page, click + to add products that can be purchased through the application.
8. When you finish adding products for in-app purchases, go to the iTunes Connect home page.
9. In iTunes Connect, click **Users and Roles**.
10. On the **Users and Roles** page, click **Sandbox Testers**.
11. On the **Sandbox Testers** page, click + to add sandbox tester accounts.

   **Important**
   
   Do not log in using this account on your device. This will deactivate the sandbox testing account and you'll need to create a new account.
12. On your device, choose **Settings**, **iTunes**. Log out of any accounts that are logged in.
13. Deploy and run the application from Xcode. When prompted, sign in with your sandbox tester account on the device.

You can now view available products, buy products, view purchased products, and more. When you buy a product, a note appears that indicates you are in the sandbox environment and will not be charged for the purchase.

**Woodland Asset Package – Legacy**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Woodland asset package includes free wilderness assets that you can use to create a forest scene or populate your own levels with vegetation and other natural features that might be found in a woodlands scene. The Woodland assets are static art assets; therefore, you will not need to recompile your project after adding these assets.

**Note**
This package is compatible with Lumberyard 1.9 and earlier.

**To install the Woodland asset package**

1. Download the Woodland Asset Package at Lumberyard Previous Versions and extract it in the `\dev\Gems\AssetCollection_Woodland` directory at the root of your Lumberyard installation. You may need to create this folder or rename the extracted folder.
2. Open the Project Configurator.
3. In the Project Configurator, under your project, click Enable Gems.
4. On the Gems (extensions) page, select Woodland Asset Collection.
5. Click Save.
6. Open Lumberyard Editor and do the following:
   - In the Rollup Bar, click Geom Entity and select your meshes.
   - Click Tools, Material Editor and select your materials and textures.
   - Click Tools, Geppetto and select your animations.

Sample images from the Woodland asset collection:
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Beach City sample project includes free assets that you can use to create your own levels. Although the Beach City sample project is intended to be a visual demo and is not a playable demo, you can add player controls to the level if you wish to make it playable.
Note
This package is compatible with Lumberyard 1.9 and earlier.

To install the Beach City sample project

1. Download the BeachCity.zip package at Lumberyard Previous Versions and extract it in your Lumberyard directory. For example, \Lumberyard\1.x.0.0.

2. Open the Lumberyard Setup Assistant (p. 16). On the Summary page, click Project Configurator.

   Note
   To ensure that the Beach City project launches, you must use Lumberyard Setup Assistant to open the Project Configurator. Lumberyard Setup Assistant copies required SDKs from the 3rdParty directory into the dev\BeachCity directory.

3. In the Project Configurator, select BeachCity.

4. Click Set as default. Close the Project Configurator.


Sample images from the Beach City sample project:
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard provides a package named Legacy Game Sample that demonstrates how to use Lumberyard features such as CryAction and CryAISystem. The fully functional game implements the necessary pieces that allow a game to communicate with some of Lumberyard's low-level systems. If you are familiar with CryEngine, the legacy game sample is a repurposed version of the GameSDK project.

You can use the legacy game sample to evaluate changes you make to the engine core and ensure your changes do not have unintended side effects. The legacy game sample is intended to be used as a reference for how to communicate with various systems of the Lumberyard engine. We do not recommend using the legacy game sample as the starting point of a new game.

**Note**
The legacy game sample is Windows only and is not supported on console or mobile devices. This sample is compatible with Lumberyard 1.9 and earlier.

**To download and access GameSDK**

1. Download the GameSDK.zip package at Lumberyard Previous Versions and extract it in your Lumberyard directory. For example, \lumberyard\1.10.0.
2. Open Lumberyard Setup Assistant (p. 16) and on the Summary page, click Project Configurator.

   **Note**
   To ensure that the GameSDK project launches, you must use Lumberyard Setup Assistant to open the Project Configurator. Lumberyard Setup Assistant copies required SDKs from the 3rdParty directory into the dev\GameSDK directory.

3. In the Project Configurator, select GameSDK.
4. Click Set as default and then close the Project Configurator.
5. Rebuild (p. 1064) the GameSDK project.
6. Open Lumberyard Setup Assistant. On the **Summary** page, click **Launch editor**.

**Note**
Audiokinetic Wave Works Interactive Sound Engine (Wwise) version 2014.1.14 or later is required to access audio for this project.
Lumberyard Editor is your primary workplace for game development. The editor provides access to the tools to create, design, and deploy your game project. You can also play the game to test your changes.

**Note**
If you receive errors from the `AssetProcessor.exe` or `AssetProcessor_temp.exe`, check for the quarantined file in your antivirus software. You can grant exceptions for the affected files.

**To open Lumberyard Editor**
- After installation, launch Lumberyard Editor with your preferred method:
  - From the desktop, double-click the Lumberyard Editor icon.
    
    **Note**
    The Lumberyard Editor desktop shortcut points to the `lumberyard_installation\dev\Bin64vc\xxx\Editor.exe` file that corresponds to the version of Visual Studio that you chose in Lumberyard Setup Assistant during setup. If you later change the version of Visual Studio that you use with Lumberyard, this desktop shortcut may no longer be valid.
  - From Lumberyard Setup Assistant, on the **Summary** page, click **Launch Editor**.
  - For Visual Studio 2017, navigate to the `lumberyard_version\dev\Bin64vc141` directory and double-click `Editor.exe`.
  - For Visual Studio 2019, navigate to the `lumberyard_version\dev\Bin64vc142` directory and double-click `Editor.exe`.

After the first launch and each time you change projects, Asset Processor runs in the background using the project configuration settings to populate the **Asset Browser**. Messages appear with status information before the editor opens. For more information about Asset Processor, see Using Asset Processor (p. 248).

**Topics**
- Lumberyard Editor Interface (p. 183)
- Using the Menu Bar (p. 183)
- Using the Top Toolbar (p. 200)
- Using the Bottom Toolbar (p. 202)
- Using Keyboard Shortcuts (p. 205)
- Using the Viewport (p. 207)
- Using the Console Window (p. 210)
- Automating the Lumberyard Editor with the Python Editor Bindings gem (p. 218)
- Demo and Video Capture (p. 230)
- Customizing Lumberyard Editor (p. 235)
- Restoring the Default Layout for Lumberyard Editor (p. 244)
Lumberyard User Guide

Lumberyard Editor Interface

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard Editor has the following panels and toolbars in the default layout:

A. Entity Outliner – The Entity Outliner lists the component entities that are included in the current level. For more information, see Entity Outliner (p. 462).

B. Asset Browser – The Asset Browser displays your assets in a tree view that mirrors your assets directory. For more information, see Asset Browser (p. 296).

C. Perspective – The 3D viewport window displays the game environment and allows you to view, create, and interact with assets. For more information, see Using the Viewport (p. 207).

D. Entity Inspector – The right pane defaults to the Entity Inspector view, which allows you to add component and entities and modify their settings and properties. For more information, see Entity Inspector (p. 475).

E. Top Toolbar – The top toolbar provides quick access to the most commonly used functions and features. You can customize the toolbar. For more information, see Using the Top Toolbar (p. 200).

F. Console – The console window allows you to enter commands to change settings for your game or execute functionality, such as connecting to a server or banning a player. The console window also displays warnings and errors for your game level, such as missing textures and models. You can access the console while in-game by pressing the tilde (~) key. For more information, see Using the Console Window (p. 210).

G. Bottom Toolbar – Use the Go to position button in the bottom toolbar to navigate to a precise X, Y, and Z location in the viewport. To transport to a point above the terrain, you can change the X, Y, and Z values to 1024, 1024, and 34, respectively. For more information, see Using the Bottom Toolbar (p. 202).

Using the Menu Bar

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The main menu bar in Lumberyard Editor provides access to the features and tools to design, run, and deploy your game, as well as, work with external tools and find online information. To access the available features and tools, you can use the menu, the buttons on the toolbars, or the keyboard. You can pause on the buttons to see tool tips. Some editors and tools have keyboard shortcuts. For a comprehensive list, see Using Keyboard Shortcuts (p. 205).

The main menu bar has the following categories of features and functionality.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Manage your resource files for your level, project, and other file management tools.</td>
</tr>
<tr>
<td>Edit</td>
<td>Select objects in your level and make changes.</td>
</tr>
<tr>
<td>Game</td>
<td>Perform operations that affect the whole game.</td>
</tr>
<tr>
<td>Tools</td>
<td>Launch a variety of editors and specialty tools.</td>
</tr>
<tr>
<td>AWS</td>
<td>Set and configure Amazon Web Services and manage your profile.</td>
</tr>
<tr>
<td>Help</td>
<td>Get information about this version of Lumberyard and how to use it.</td>
</tr>
</tbody>
</table>

File

In the File menu, you can manage your game project such as opening and saving level files, show a log file, and modify your project's settings.
<table>
<thead>
<tr>
<th>File Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Create a new level. For more information, see Creating a New Level (p. 1219).</td>
</tr>
<tr>
<td>Open</td>
<td>Switch to another level in your project.</td>
</tr>
<tr>
<td>Open Recent</td>
<td>Open a recently opened level.</td>
</tr>
<tr>
<td>Save</td>
<td>Update the level file with your changes.</td>
</tr>
<tr>
<td>Save as</td>
<td>Create a copy of the open level with a new name.</td>
</tr>
<tr>
<td>Save Level Resources</td>
<td>Update all of the assets with changes made in the open level.</td>
</tr>
<tr>
<td>Save Level Statistics</td>
<td>Save statistics for the open level to an .xml file.</td>
</tr>
<tr>
<td>Save Modified External Layers</td>
<td>Save only the external layers that have been modified since the last save.</td>
</tr>
<tr>
<td>Project Settings</td>
<td>Launch other file management tools:</td>
</tr>
<tr>
<td></td>
<td>• Switch Projects – Opens the Project Configurator</td>
</tr>
<tr>
<td></td>
<td>• Configure Gems – Opens the Project Configurator</td>
</tr>
<tr>
<td></td>
<td>• Deployment Tool – Opens the editor and console for selecting a deployment platform and to use a console.</td>
</tr>
<tr>
<td>Show Log File</td>
<td>Show the log file that contains all text printed in the console to a .log file in the project's directory.</td>
</tr>
<tr>
<td>Upgrade Legacy Entities</td>
<td>Launch the Legacy Converter tool to convert your legacy entities to components that you can edit in Lumberyard Editor. For more information, see Converting Entities with the Legacy Converter (p. 963).</td>
</tr>
<tr>
<td>Exit</td>
<td>Close Lumberyard Editor. At the prompt, you can save any changes made since the last save was performed.</td>
</tr>
</tbody>
</table>

**Edit**

In the **Edit** menu, you can select and manipulate objects, such as undo and redo actions, and select and hide objects.
<table>
<thead>
<tr>
<th>Edit Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Revert the last action.</td>
</tr>
<tr>
<td>Redo</td>
<td>Apply the last action.</td>
</tr>
<tr>
<td>Duplicate</td>
<td>Create a copy of the selected object</td>
</tr>
<tr>
<td>Delete</td>
<td>Remove the selected object from the level and its base object remains available in the Asset Browser.</td>
</tr>
<tr>
<td>Select All</td>
<td>Specify that all visible, unlocked objects are selected for changing or applying settings.</td>
</tr>
<tr>
<td>Deselect All</td>
<td>Remove focus from the objects that are currently selected.</td>
</tr>
<tr>
<td>Next Selection Mask</td>
<td>Select the next selection mask.</td>
</tr>
<tr>
<td>Invert Selection</td>
<td>Swap the selection set from the currently, selected set of objects to the other available objects.</td>
</tr>
<tr>
<td>Hide Selection</td>
<td>Set the selected object as invisible.</td>
</tr>
<tr>
<td>Show Selection</td>
<td>Set the selected object as visible.</td>
</tr>
<tr>
<td>Show Last Hidden</td>
<td>Reverse the visibility setting of the hidden object last modified.</td>
</tr>
<tr>
<td>Unhide All</td>
<td>Set all invisible objects as visible.</td>
</tr>
</tbody>
</table>
### Edit Menu Item

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Specify multiple selected objects as a set to manipulate and modify. The viewport displays a green box around the objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modify</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify</td>
<td>Show a submenu of actions. See the Modify (p. 188) section.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lock Selection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock Selection</td>
<td>Set the selected object as frozen. Frozen objects are uneditable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unlock All</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlock All</td>
<td>Set all frozen objects as editable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rename Object(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename Object(s)</td>
<td>Opens a dialog box to specify a new name. If you rename multiple objects at once, each object is appended with a number.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set Object(s) Height</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Object(s) Height</td>
<td>Opens a dialog box to specify a specified height (in meters) above the terrain.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Editor Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor Settings</td>
<td>See the Editor Settings (p. 189) section.</td>
</tr>
</tbody>
</table>

### Group

In the **Group** menu, you can apply changes to multiple objects as a set, such as grouping and ungrouping objects.

<table>
<thead>
<tr>
<th>Group Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Specify multiple objects as a set to manipulate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ungroup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ungroup</td>
<td>Separate all of the objects that are in the selected group.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Group</td>
<td>Allow an object within a set to be selected and manipulated independently of the group.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attach to Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach to Group</td>
<td>Add selected objects to a group.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Detach from Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detach from Group</td>
<td>Remove selected objects from a group.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hold</td>
<td>Save the current state for the group temporarily, so that you can test and experiment with alternative settings. You can keep the new settings or click <strong>Fetch</strong> to undo them.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Restores the state of the group to the saved state specified with <strong>Hold</strong>. This option is only available if you placed a <strong>Hold</strong> at some point.</td>
</tr>
</tbody>
</table>
Modify

In the Modify menu, you can manipulate attributes and properties of objects and entities, such as height, alignment, and material.

<table>
<thead>
<tr>
<th>Modify Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>Create hierarchies between objects.</td>
</tr>
<tr>
<td>Unlink</td>
<td>Remove the connection between linked objects.</td>
</tr>
<tr>
<td>Align</td>
<td>Display a submenu with options for placing an object in relationship to the grid, to another object, or to the selected surface, which moves the pivot point of the object.</td>
</tr>
<tr>
<td></td>
<td>If you align an object to another object that has modified scale or rotation, the original object will use the modified settings along with the position data. To override this action, use the following keys (single or combination) when you select the original object:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Ctrl</strong> – Align the object to the bounding box.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Alt</strong> – Use the object’s existing rotation.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Shift</strong> – Use the object’s existing scale.</td>
</tr>
<tr>
<td>Constrain</td>
<td>Limit the movement of an object to the XYZ axes, XY planes, or to the surface of the terrain and objects.</td>
</tr>
<tr>
<td>Snap</td>
<td>Place an object on the grid or a rotational increment.</td>
</tr>
<tr>
<td>Transform Mode</td>
<td>Displays a submenu to apply the following changes to an object:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Select Mode</strong> – Select an object to modify.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Move</strong> – Change the location of an object along the axes.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Rotate</strong> – Select and spin an object on the selected axis.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Scale</strong> – Select and resize an object proportionally.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Select Terrain</strong> – Switch from the selected object to the terrain.</td>
</tr>
<tr>
<td>Convert to</td>
<td>Change the selected object to another type of entity: brush, geometry entity, designer object, static entity, game volume,</td>
</tr>
</tbody>
</table>
Modify Menu Item | Description
--- | ---
| or component entity. For more information, see the Converting Entities with the Legacy Converter (p. 963). |

Fast Rotate | Quickly spin the selected object on the specified axis or with the degree value that you set for Rotate Angle. |

Sub Object Mode | If the Edit Mesh function is enabled, select and edit the geometry components of the object. |

Save Object(s) | Update the file with the changes made to the selected objects. |

Load Object(s) | Open the dialog box to browse and select objects from the game directory. |

Update Procedural Vegetation | Reapply settings in the .sbsar files. |

**Editor Settings**

In the Editor Settings menu, you can customize your editing experience.

---

<table>
<thead>
<tr>
<th>Editor Settings Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Preferences</td>
<td>Modify the global settings for Lumberyard Editor and its tools. For more information, see Changing Preferences (p. 238).</td>
</tr>
<tr>
<td>Graphics Performance</td>
<td>Select your preferred display setting:</td>
</tr>
<tr>
<td></td>
<td>• PC – Enable specific resolution display settings (Note: Some values are DX11 specific):</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• OSX GL osx-gl.cfg – same OSX Metal osx-metal.cfg.</td>
</tr>
<tr>
<td></td>
<td>• OSX Metal – Emulate the macOS display settings that use OpenGL graphics.</td>
</tr>
<tr>
<td></td>
<td>• Android – Enable specific resolution display settings.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• iOS – Enable specific resolution display settings.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Apple TV – Emulate the Apple TV display settings.</td>
</tr>
<tr>
<td>Keyboard Customization</td>
<td>Configure toolbars, menus, and keyboard shortcuts. You can customize your keyboard settings, with the option to import or export saved custom keyboard settings.</td>
</tr>
</tbody>
</table>
Game

In the Game menu, you can add and test enhancements to your game project.

<table>
<thead>
<tr>
<th>Game Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Play Game</strong></td>
<td>Switch to game mode (runtime). To return to edit mode, press Esc.</td>
</tr>
<tr>
<td><strong>Enable Physics/AI</strong></td>
<td>Toggle physics and AI in your level to use the PhysX system if the event bus and handlers are not specified. For more information, see Working with the Event Bus (EBus) system (p. 1851).</td>
</tr>
<tr>
<td><strong>Export to Engine</strong></td>
<td>Export the level data to a level.pak file in order to play the level in game mode.</td>
</tr>
<tr>
<td><strong>Export Selected Objects</strong></td>
<td>Save the selected geometry to an .obj or .fbx file to make it available for use with other entities.</td>
</tr>
<tr>
<td><strong>Export Occlusion Mesh</strong></td>
<td>Save the occlusion mesh for application to other entities.</td>
</tr>
<tr>
<td><strong>Terrain Collision</strong></td>
<td>Toggle the ability of the camera to move with the terrain surface and prevent flying under the terrain surface.</td>
</tr>
<tr>
<td><strong>Edit Equipment Packs</strong></td>
<td>Open the Edit Equipment Packs window to add, delete, rename, or reorder the equipment for an actor.</td>
</tr>
<tr>
<td><strong>Toggle SP/MP GameRules</strong></td>
<td>Toggle between single player and multiplayer game rules.</td>
</tr>
<tr>
<td><strong>Synchronize Player with Camera</strong></td>
<td>Set the player's position relative to the camera position.</td>
</tr>
<tr>
<td><strong>AI</strong></td>
<td>Submenu for artificial intelligence options. See the AI (p. 191) section.</td>
</tr>
<tr>
<td><strong>Audio</strong></td>
<td>Submenu for sound options. See the Audio (p. 192) section.</td>
</tr>
<tr>
<td>Game Menu Item</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Clouds</td>
<td>Submenu enabled when the Sky Clouds gem is included in the project. See the <a href="p.193">Clouds (p. 193)</a> section.</td>
</tr>
<tr>
<td>Prefabs</td>
<td>Submenu for the legacy version of slices. See the <a href="p.193">Prefabs (p. 193)</a> section.</td>
</tr>
<tr>
<td>Terrain</td>
<td>Submenu for managing the terrain. See the <a href="p.194">Terrain (p. 194)</a> section.</td>
</tr>
<tr>
<td>Debugging</td>
<td>Submenu offer options for reloading specific files and for error checking. See the <a href="p.196">Tools (p. 196)</a> section.</td>
</tr>
</tbody>
</table>

**AI**

In the **AI** menu, you can manage AI settings such as generating AI navigation and updating the AI system within a level. For more information, see the [Navigation Area (p. 693)](p.693) component.

<table>
<thead>
<tr>
<th>AI Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate All AI</td>
<td>Generates the AI navigation to include triangulation, 3D navigation volumes, flight navigation, and waypoints.</td>
</tr>
<tr>
<td>Generate Triangulation</td>
<td>Generates only the triangulation of the navigation mesh that is used for outdoor levels.</td>
</tr>
<tr>
<td>Generate 3D Navigation Volumes</td>
<td>Generates only the 3D navigation data for 3D volumes that are used by alien AI agents; volumes are defined by AINavigationModifier and a Volume navigation type.</td>
</tr>
<tr>
<td>AI Menu Item</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Generate Flight Navigation</td>
<td>Generates only the 2.5D navigation data for volumes used by flying AI agents; volumes are defined by AINavigationModifier and a Flight navigation type.</td>
</tr>
<tr>
<td>Generate Waypoints</td>
<td>Generates only the links for indoor waypoints.</td>
</tr>
<tr>
<td>Validate Navigation</td>
<td>Check the generated data for various problems and display warnings if any problems are found (for example, bad object placement, overlapping forbidden areas, or corruptions).</td>
</tr>
<tr>
<td>Clear All Navigation</td>
<td>Remove the triangulation, 3D navigation volumes, flight navigation, and waypoint data from the level.</td>
</tr>
<tr>
<td>Generate Spawner Entity Code</td>
<td>Search for AI entity classes and generate an .ent file for each; associate an entity class name with the Lua base file for that entity.</td>
</tr>
<tr>
<td>Generate 3D Debug Voxels</td>
<td>Generate debugging information for volume navigation regions when the ai_DebugDraw console variable is enabled.</td>
</tr>
<tr>
<td>Create New Navigation Area</td>
<td>Create a navigation area.</td>
</tr>
<tr>
<td>Request a full MNM rebuild</td>
<td>Perform a full rebuild of all multi-layer navigation mesh (MNM) data. Use this periodically if the Continuous Update feature is off (see below).</td>
</tr>
<tr>
<td>Show Navigation Areas</td>
<td>Enable blue areas to highlight MNM areas on the terrain in the viewport.</td>
</tr>
<tr>
<td>Add Navigation Seed</td>
<td>Add an entity designated as points of off-mesh accessibility of an MNM.</td>
</tr>
<tr>
<td>Continuous Update</td>
<td>Enable automatic updates to the MNM data. If disabled, you need to select Request a full MNM rebuild in order for the mesh data to update.</td>
</tr>
<tr>
<td>Visualize Navigation Accessibility</td>
<td>Display areas that are inaccessible to the AI agent in red and accessible areas in blue in the viewport.</td>
</tr>
<tr>
<td>View Agent Type</td>
<td>Select from a submenu of agent types such as MediumSizedCharacters. You can add agent types to the navigation.xml file in the lumberyard_version\dev \your_project\scripts directory.</td>
</tr>
<tr>
<td>Generate Cover Surfaces</td>
<td>Generate the data specifying areas available for hiding and avoiding danger.</td>
</tr>
<tr>
<td>AIPoint Pick Link</td>
<td>Combine AI navigation modifier points.</td>
</tr>
<tr>
<td>AIPoint Pick Impass Link</td>
<td>Restricts AI navigation modifier points to prevent AI from walking on the points.</td>
</tr>
</tbody>
</table>

**Audio**

In the **Audio** menu, you can stop all sounds or refresh the audio.
### Audio Menu Item

<table>
<thead>
<tr>
<th>Audio Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop All Sounds</td>
<td>Silence all sounds in the level.</td>
</tr>
<tr>
<td>Refresh Audio</td>
<td>Reapply settings to all of the sounds in the level.</td>
</tr>
</tbody>
</table>

### Clouds

**Clouds**

In the **Clouds** menu, you can create, open, close, and delete your custom cloud assets. The project must have the *Sky Clouds Gem* (p. 1202) enabled. For more information, see *Sky Clouds Gem* (p. 1202).

<table>
<thead>
<tr>
<th>Clouds Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create a new cloud asset.</td>
</tr>
<tr>
<td>Destroy</td>
<td>Remove a custom cloud asset.</td>
</tr>
<tr>
<td>Open</td>
<td>Open the selected cloud asset.</td>
</tr>
<tr>
<td>Close</td>
<td>Close the selected cloud asset.</td>
</tr>
</tbody>
</table>

### Prefabs

**Prefabs**

In the **Prefabs** menu, you can modify CryEntities defined in the prefab library.

**Note**

We recommend that you use slices instead. For more information, see *Working with Slices* (p. 510).

<table>
<thead>
<tr>
<th>Prefabs Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Prefab from Selected Object(s)</td>
<td>Create a prefab from selected objects.</td>
</tr>
<tr>
<td>Add Selected Object(s) to Prefab</td>
<td>Add the selected objects to the prefab.</td>
</tr>
<tr>
<td>Clone Selected Object(s)</td>
<td>Clone the selected objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prefabs Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Prefab from Selected Object(s)</td>
<td>Create a prefab from selected objects.</td>
</tr>
<tr>
<td>Add Selected Object(s) to Prefab</td>
<td>Add the selected objects to the prefab.</td>
</tr>
<tr>
<td>Clone Selected Object(s)</td>
<td>Clone the selected objects.</td>
</tr>
<tr>
<td>Prefabs Menu Item</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Extract Selected Object(s)</td>
<td>Extract the selected objects from the prefab.</td>
</tr>
<tr>
<td>Open All</td>
<td>Open all prefabs.</td>
</tr>
<tr>
<td>Close All</td>
<td>Close all prefabs.</td>
</tr>
<tr>
<td>Reload All</td>
<td>Reload all prefabs.</td>
</tr>
</tbody>
</table>

### Terrain

In the **Terrain** menu, you can specify changes that affect the game world and terrain appearance.

<table>
<thead>
<tr>
<th>Terrain Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Terrain Texture</td>
<td>Generate the terrain surface texture in a compressed format into the <code>terraintexture.pak</code> file; you must do this in order for changes made by the terrain painter to be visible in game mode.</td>
</tr>
<tr>
<td>Generate Terrain</td>
<td>Open the <strong>Terrain Editor</strong> to generate terrain for the level.</td>
</tr>
<tr>
<td>Edit Terrain</td>
<td>Open the <strong>Terrain Editor</strong> to modify the terrain settings.</td>
</tr>
<tr>
<td>Export/Import Megaterrain Texture</td>
<td>Save or include a megaterrain texture, which is a diffuse texture that provides changes in detail as the camera moves closer to the terrain.</td>
</tr>
<tr>
<td>Export Terrain Block</td>
<td>Save a section of the terrain to a terrain block <code>.trb</code> file.</td>
</tr>
<tr>
<td>Import Terrain Block</td>
<td>Include terrain from a saved <code>.trb</code> file.</td>
</tr>
<tr>
<td>Resize Terrain</td>
<td>Open the <strong>Terrain Resize</strong> tool to modify the terrain size.</td>
</tr>
<tr>
<td>Terrain Modify</td>
<td>Flatten or smooth the terrain.</td>
</tr>
<tr>
<td>Edit Vegetation</td>
<td>Open the <strong>Vegetation</strong> section on the <strong>Terrain</strong> tab in the <strong>Rollup Bar</strong> in order to modify the vegetation.</td>
</tr>
</tbody>
</table>
### Terrain Menu Item

<table>
<thead>
<tr>
<th>Terrain Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paint Layers</td>
<td>Open the <strong>Layer Painter</strong> section on the <strong>Terrain</strong> tab in the <strong>Rollup Bar</strong> in order to modify settings on the layer.</td>
</tr>
<tr>
<td>Refine Terrain Texture Tiles</td>
<td>Divide the terrain tiles into smaller sections.</td>
</tr>
<tr>
<td>Export Terrain Area</td>
<td>Save the selected terrain area to an <code>.obj</code> or <code>.fbx</code> file.</td>
</tr>
<tr>
<td>Export Terrain Area with Objects</td>
<td>Save the selected terrain area and all objects within to an <code>.obj</code> or <code>.fbx</code> file.</td>
</tr>
</tbody>
</table>

### Debugging

In the **Debugging** menu, you can reload scripts, textures, geometry, and terrain. Other debugging options include configuring user commands and checking the level for errors.

#### Debugging Menu Item

<table>
<thead>
<tr>
<th>Debugging Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reload Scripts (LEGACY)</td>
<td>Reapply all scripts or apply scripts separately for actor, AI, entity, item, and UI.</td>
</tr>
<tr>
<td>Reload Textures/Shaders</td>
<td>Reapply the settings for all of the textures and shaders used in the level.</td>
</tr>
<tr>
<td>Reload Geometry</td>
<td>Reapply the settings for all of the geometry used in the level.</td>
</tr>
<tr>
<td>Reload Terrain</td>
<td>Reapply the settings for the selected terrain.</td>
</tr>
<tr>
<td>Resolve Missing Objects/Materials...</td>
<td>Check the level and resolve all object and material issues.</td>
</tr>
<tr>
<td>Enable File Change Monitoring</td>
<td>Enable tracking to detect changes to files.</td>
</tr>
<tr>
<td>Check Object Positions</td>
<td>Check the position of all objects in the level.</td>
</tr>
<tr>
<td>Debugging Menu Item</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Clear Registry Data</td>
<td>Remove the registry data stored for all custom toolbars.</td>
</tr>
<tr>
<td>Check Level for Errors</td>
<td>Compile a list of errors within the level (such as duplicate objects and missing assets) and display it in the console window.</td>
</tr>
<tr>
<td>Save Level Statistics</td>
<td>Update the <code>your_level_name.xml</code> file in the <code>C:\Amazon\Lumberyard\your_version\dev\Cache\your_game\pc\your_game\testresults</code> directory with current data.</td>
</tr>
<tr>
<td>Compile Script</td>
<td>Compile an entity script.</td>
</tr>
<tr>
<td>Reduce Working Set</td>
<td>Reduce memory consumption.</td>
</tr>
<tr>
<td>Update Procedural Vegetation</td>
<td>Apply modifications made to the procedural vegetation.</td>
</tr>
<tr>
<td>Configure Toolbox Macros</td>
<td>Open the <code>Tools Configuration</code> window to create shortcuts for the console commands.</td>
</tr>
<tr>
<td>Toolbox Macros</td>
<td>Display the shortcuts in the console and Lumberyard Editor commands that you created.</td>
</tr>
<tr>
<td>Script Help</td>
<td>Open the <code>Script Help</code> window to view a list of commands, descriptions, and examples.</td>
</tr>
</tbody>
</table>

**Tools**

In the **Tools** menu, you can access Lumberyard Editor tools and plugins. For more information, see [Lumberyard Editors and Tools (p. 32)](p. 32).
In the **View** menu, you can customize the layout and the viewport of Lumberyard Editor.

<table>
<thead>
<tr>
<th>View Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Center on Selection</strong></td>
<td>Select an object to zoom to its boundaries. You can then press <strong>Alt</strong> and use the mouse to pan around the object, which remains centered on the screen.</td>
</tr>
<tr>
<td><strong>Show Quick Access Bar</strong></td>
<td>Show or hide the quick access bar.</td>
</tr>
<tr>
<td><strong>Error report</strong></td>
<td>Open the file with details about errors.</td>
</tr>
</tbody>
</table>
### View Menu Item

<table>
<thead>
<tr>
<th>View Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Layouts</strong></td>
<td>Choose or save the pane layout shown in Lumberyard Editor:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Component Entity Layout</strong> – Display the <strong>Entity Outliner</strong>, <strong>Entity Inspector</strong>, and <strong>Asset Browser</strong> with the viewport,</td>
</tr>
<tr>
<td></td>
<td>• <strong>Legacy Layout</strong> – Display the <strong>Rollup Bar</strong> and viewport only.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Save Layout</strong> – Save your current layout.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Restore Default Layout</strong> – Restore the layout to the default (<strong>Component Entity Layout</strong>).</td>
</tr>
<tr>
<td></td>
<td>For more information, see <a href="#">Restoring the Default Layout for Lumberyard Editor</a> (p. 244).</td>
</tr>
</tbody>
</table>

| **Viewport**   | Submenu for the **Perspective** panel. See the [Viewport](#) (p. 198) section. |

<table>
<thead>
<tr>
<th><strong>Viewport Menu Item</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wireframe</strong></td>
<td>Enable or disable the wireframe view.</td>
</tr>
<tr>
<td><strong>Ruler</strong></td>
<td>Add a tool measure the distance from one point to another.</td>
</tr>
<tr>
<td><strong>Grid Settings</strong></td>
<td>Set grid line spacing, angle snapping, and rotation and translation settings.</td>
</tr>
<tr>
<td><strong>Configure Layout</strong></td>
<td>Select a preconfigured layout.</td>
</tr>
<tr>
<td><strong>Goto Coordinates</strong></td>
<td>Specify the camera position in XYZ coordinates and move the camera to that position.</td>
</tr>
<tr>
<td><strong>Center on Selection</strong></td>
<td>Go to the currently selected object in the viewport.</td>
</tr>
</tbody>
</table>
Viewport Menu Item | Description
--- | ---
Goto Location | Go to one of 10 predefined locations in the viewport.
Remember Location | Save up to 10 locations in the viewport.
Change Move Speed | Change the movement speed for all objects in the level.
Switch Camera | Change the camera for the viewport:
  - Default Camera – Select the default camera.
  - Sequence Camera – Select the camera that is used in a track view sequence.
  - Selected Camera Object – Select the camera entity.
  - Cycle Camera – Select the next camera.
Show/Hide Helpers | Show or hide all helper objects.

AWS

In the AWS menu, you can sign up for an Amazon Web Services (AWS) account and set up services using Cloud Canvas and Amazon GameLift.

AWS Menu Item | Description
--- | ---
Credentials manager | Add or edit an AWS profile.
Cloud Canvas | Select a deployment or see more information in Understanding Cloud Canvas Resource Manager (p. 2110) or Using Dynamic Content Manager (p. 2155).
Commerce | Learn how to submit your game to Amazon's Digital Software store using Merch by Amazon or Publishing on Amazon.
GameLift | Access the Amazon GameLift console or learn more about the game server hosting and matchmaking solution built on AWS.
Open AWS Console | Open the AWS Management Console and access Amazon Cognito, user management services; Amazon DynamoDB, NoSQL database service; Amazon S3, cloud storage; and AWS Lambda, serverless computing.

Help

In the Help menu, you can search the Lumberyard documentation, open Lumberyard tutorials and resources, and view information about Lumberyard Editor.
Using the Top Toolbar

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Lumberyard Editor provides a toolbar that allows you to easily access various editor tools and features. The toolbar is docked at the top of the editor by default, but you can also dock it vertically on the edges of the editor or undock it from the editor. To customize the toolbar, right-click anywhere on the toolbar and select Customize from the context menu. You can choose which toolbars, views, or modes to include. You can also add commands to a toolbar.

You can access Lumberyard tools and features using one of the following methods:

- Toolbar buttons
- Tools menu
- Keyboard shortcuts (p. 205)

For information about the bottom toolbar, see Using the Bottom Toolbar (p. 202).

**EditMode Toolbar**

The **EditMode** toolbar includes various tools for general level editing:

- A – Revert or apply the last command
- B – Link or unlink the selected object
- C – Filter what you can select in the viewport: all, brushes, no brushes, entities, prefabs, areas, shapes, AI points, decals, solids, or no solids
- D – Use the translation tools to select, move, rotate, or scale an object or object type; and select or rotate a terrain area
- E – Select the reference coordinate system
- F – Specify the axis constraint by locking on the x-, y-, or z-axis or xy-plane
- G – Use the object placement tools to follow the terrain, snap to objects, snap to grid, snap to angle, or show the ruler
- H – Use the asset organization tools to open the object selector, create a selection, delete a selection, save selected objects, or load selected objects
- I – Select the current layer

**Object Toolbar**

The **Object** toolbar includes various tools for object alignment and manipulation:

- A – Go to the selected object
- B – Align the selection to an object by choosing the source object, clicking the tool, and then clicking the target object
- C – Align the object to the grid
- D – Set the object's height
- E – Align the object to the terrain surface normal (press and hold Ctrl for object surface normal alignment)
- F – Freeze or unfreeze the selected object
- G – Apply vertex snapping for the selected object
Editors Toolbar

The Editors toolbar allows you to access various editor tools:

- **A** – Open the Asset Browser
- **B** – Open the Layer Editor
- **C** – Open the LOD Generator
- **D** – Open the Material Editor
- **E** – Opens Geppetto
- **F** – Open the Mannequin Editor
- **H** – Open the AI Debugger
- **I** – Open the Track View
- **J** – Open the Audio Controls Editor
- **K** – Open the Terrain Editor
- **L** – Open the Terrain Texture Layers editor
- **M** – Open the Particle Editor
- **N** – Open the Time of Day editor
- **O** – Open the Sun Trajectory Tool
- **P** – Open the Database View
- **Q** – Open the UI Editor

Substance Toolbar

The Substance toolbar includes a button that opens the Substance Editor.

Using the Bottom Toolbar

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Lumberyard Editor includes a bottom toolbar that provides status as well as the features below.

**Status**

The *status* bar (A) displays the number of selected entities and provides functional hints for buttons or menu items in Lumberyard Editor.

**Coordinates/Transforms**

The *coordinates/transform* area (B) shows the position of the cursor or the status of a transform, and allows you to enter new transform values. The information in these fields vary based on your tasks:

- When creating an entity or moving the mouse in the viewport, these fields show the cursor location in absolute world coordinates.
- When translating an entity by dragging it in the viewport, these fields show entity's location in absolute world coordinates.
- While transforming an entity, these fields change to spinners in which you can directly type values, or modify by scrolling the mouse wheel.
- While a transform button is active and multiple entities are selected, these fields show the first selection's transform values.

**Set Vector**

The *Set Vector* button (C) allows you to set the position, rotation or scale for the selected entities.

**Goto Position**

The *Goto Position* button (D) opens the *Go to position* dialog box to set a world position and rotation for the perspective view. You can enter floating point coordinates and rotation, or use the spinners to specify values. Clicking the *Go To* button immediately moves the viewport to the specified coordinates and rotates the view.
Lock Selection

The Lock Selection button (E) toggles selection locking, preventing you from inadvertently selecting something else in a level.

When your selection is locked, you can click or drag the mouse anywhere in the viewport without losing your selection. To deselect or alter your selection, click Lock Selection again to unlock the selection.

Lock Axis Vectors

The Lock Axis Vectors button (F) makes scale operations uniform.

Speed Control

The Speed button (G) allows you to change the speed of viewport movement. Select a preset speed from the list, enter a value into the field, or use the spinners to adjust the speed up or down.

Terrain Collision

The Enable Terrain Camera Collision button (H) toggles terrain collision to inhibit camera movement below the terrain surface.

Move Player and Camera Separately

The Move Player and Camera Separately button (I) allows the viewport camera to be moved separately from the player entity.

Simulate

The Simulate button (J) toggles physics simulation and AI, allowing you to test physics and AI behavior directly in the editor without entering game mode.
Mute Audio

The Mute Audio button (K) mutes audio in both editor and play mode.

VR Preview

The VR Preview button (L) previews your project in virtual reality mode (p. 3258) when a virtual reality (p. 3248) gem is enabled.

Using Keyboard Shortcuts

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard supports the following keyboard shortcuts.

File Menu

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+O</td>
<td>Open a level</td>
</tr>
<tr>
<td>Ctrl+S</td>
<td>Save the level</td>
</tr>
<tr>
<td>Ctrl+Alt+F</td>
<td>Restore the saved state</td>
</tr>
<tr>
<td>Ctrl+Alt+H</td>
<td>Save the current state</td>
</tr>
<tr>
<td>Ctrl+Shift+S</td>
<td>Save the selected object</td>
</tr>
</tbody>
</table>

Edit Menu

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Toggle snap-to-grid</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Duplicate the selected object</td>
</tr>
<tr>
<td>Ctrl+Z</td>
<td>Undo the last operation</td>
</tr>
<tr>
<td>Ctrl+Shift+Z</td>
<td>Redo the last operation</td>
</tr>
<tr>
<td>Ctrl+Shift+Space</td>
<td>Lock the selection</td>
</tr>
</tbody>
</table>

Game Menu

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+E</td>
<td>Export the level</td>
</tr>
<tr>
<td>Ctrl+G</td>
<td>Enter game mode (Esc to exit)</td>
</tr>
</tbody>
</table>
Using Keyboard Shortcuts

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+P</td>
<td>Enable AI or physics</td>
</tr>
</tbody>
</table>

**Tools Menu**

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Open the Material Editor</td>
</tr>
<tr>
<td>`</td>
<td>Open the console window</td>
</tr>
<tr>
<td>[</td>
<td>Increase the brush radius size</td>
</tr>
<tr>
<td>]</td>
<td>Decrease the brush radius size</td>
</tr>
<tr>
<td>Shift+[</td>
<td>Decrease the hardness shape of the fall-off curve between the inner and outer radius of the brush</td>
</tr>
<tr>
<td>Shift+]</td>
<td>Increase the hardness shape of the fall-off curve between the inner and outer radius of the brush</td>
</tr>
</tbody>
</table>

**Select Toolbar**

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Select the object</td>
</tr>
<tr>
<td>2</td>
<td>Select and move the object</td>
</tr>
<tr>
<td>3</td>
<td>Select and rotate the object</td>
</tr>
<tr>
<td>4</td>
<td>Select and scale the object</td>
</tr>
<tr>
<td>5</td>
<td>Select the terrain area</td>
</tr>
</tbody>
</table>

**Hide Toolbar**

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Freeze the selected object</td>
</tr>
<tr>
<td>H</td>
<td>Hide the selected object</td>
</tr>
<tr>
<td>Ctrl+F</td>
<td>Unfreeze all objects</td>
</tr>
<tr>
<td>Ctrl+H</td>
<td>Unhide all hidden objects</td>
</tr>
</tbody>
</table>

**Constraint Toolbar**

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+1</td>
<td>Follow the terrain</td>
</tr>
<tr>
<td>Ctrl+2</td>
<td>Lock on the xy-plane</td>
</tr>
<tr>
<td>Ctrl+3</td>
<td>Lock on the x-axis</td>
</tr>
</tbody>
</table>
Using the Viewport

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+4</td>
<td>Lock on the y-axis</td>
</tr>
<tr>
<td>Ctrl+5</td>
<td>Lock on the z-axis</td>
</tr>
</tbody>
</table>

### Perspective Viewport

<table>
<thead>
<tr>
<th>Keyboard shortcut</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>Move forward in the viewport</td>
</tr>
<tr>
<td>A</td>
<td>Move backward in the viewport</td>
</tr>
<tr>
<td>S</td>
<td>Move left in the viewport</td>
</tr>
<tr>
<td>D</td>
<td>Move right in the viewport</td>
</tr>
<tr>
<td>Q</td>
<td>Move up in the viewport</td>
</tr>
<tr>
<td>E</td>
<td>Move down in the viewport</td>
</tr>
<tr>
<td>Z</td>
<td>Focus the camera to the selected object in the viewport</td>
</tr>
<tr>
<td>F3</td>
<td>Toggle the wireframe view</td>
</tr>
<tr>
<td>Alt+middle mouse button</td>
<td>Rotate around the selected object</td>
</tr>
<tr>
<td>Alt+right click</td>
<td>Zoom in and out</td>
</tr>
<tr>
<td>Ctrl+F1 (or F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12)</td>
<td>Save the viewport location</td>
</tr>
<tr>
<td>Shift+F1 (or F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12)</td>
<td>Move to the saved viewport location</td>
</tr>
<tr>
<td>Shift+Spacebar</td>
<td>Show or hide helpers</td>
</tr>
</tbody>
</table>

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com玩游戏/3DE) or visit the [AWS Game Tech blog](https://aws.amazon.com/game-tech/) to learn more.

The viewport window (called Perspective in Lumberyard Editor) displays the scene that is rendered by the engine. The majority of level design occurs in the viewport, including object placement, terrain editing, in-editor play testing, and asset creation and interaction. You can also use dynamic and flexible tools to understand the 3D relationships among objects in a level.

The Perspective header includes a search box, field of view (FOV), screen ratio information, and options to show or hide debug information.

**Topics**
- Changing the Field of View (FOV) (p. 208)
- Changing the Camera View (p. 209)
To customize the viewport

1. In Lumberyard Editor, in the viewport title bar, right-click **Perspective** and choose **Configure Layout**.
2. In the **Layout Configuration** dialog box, select your preferred layout.
3. Click **OK** to save your changes.

### Changing the Field of View (FOV)

You can change the default camera's FOV for the Lumberyard Editor viewport or the FOV for your game's camera. Follow the steps to change the default camera's FOV. To change the FOV for your game's camera, see **Camera** (p. 566).

#### To change the default camera's FOV

1. In Lumberyard Editor, choose **Edit, Editor Settings, Global Preferences**.
2. In the **Preferences** window, under **Viewports**, click **General**.
3. In the right pane, under **General Viewport Settings**, for **Perspective View FOV**, set the desired value in degrees.
4. Click **OK** to save your changes.
Changing the Camera View

You can use the Viewport Camera Selector to quickly position and orient a camera in your game. You can choose between all in-game cameras and the editor camera. When you possess the camera, you can use the editor controls to manipulate the camera.

To possess the camera and move around

1. In Lumberyard Editor, in the Perspective viewport, select an entity with a Camera component or create one if it doesn't exist.
2. In the Entity Inspector, under Camera, choose Be this camera. This allows the editor to use the selected camera as its view.

Note
After choosing Be this camera, moving the view in the editor changes the transform for the camera entity. As you look around in the viewport while possessing a camera, the corresponding position and orientation for the transform is updated accordingly.
3. Do any of the following:
   - Change to the default view: In the Entity Inspector, under Camera, choose Return to default editor camera.
   - Choose another camera to possess in the Viewport Camera Selector.
   - Disable the Viewport Camera Selector controls by entering game mode (Ctrl+G).

Using the Console Window

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In Lumberyard Editor, the console window shows a running list of all editor commands, processes, and output. For example, when you delete an entity, the console shows the action and the output. You can use the console to enter or modify console variables (CVARs). Console variables are a type of variable that you can manipulate in Lumberyard Editor.

**Topics**
- Viewing the Console Window (p. 210)
- Exporting All Console Commands and Variables (p. 211)
- Configuring Console Variables (p. 212)
- Console in Depth (p. 213)
- Console Variable Tutorial (p. 216)

**Viewing the Console Window**

You can enter commands directly in the console window or search and edit console variables.

**To view the console window**

1. In Lumberyard Editor, choose Tools, Console.
2. Click the X icon in the left corner to open the Console Variables window. The Console Variables window lists all available console variables.
3. Enter a name to search for a specific console variable. To learn more about the console variable, pause on the name.

Exporting All Console Commands and Variables

You can retrieve a complete list of console commands and variables, including their descriptions and assigned values.

To export a list of all console commands and variables

1. In the Console window, enter the following command: DumpCommandsVars
2. Navigate to the lumberyard_version\dev\ directory and then open the consolecommandandvars.txt file.

Example

You can see the available commands and variables in the file.
3. You can specify a sub-string parameter to restrict the results that you want. For example, the `DumpCommandsVars i_` command exports all commands and variables that begin with the prefix `i_`, such as `i_giveallitems` and `i_debug_projectiles`.

### Configuring Console Variables

Console variables can also be set in code or specified in configuration files. Console variables are executed in the following order:

1. Configuration files:
   - The `game.cfg` file in your project directory
   - The `lumberyard_version\dev\system_gamesystem.cfg` file for your game system
   - The `lumberyard_version\dev\engine\config\user.cfg` file
   - The `level.cfg` file in your project's level directory

2. Code

3. Console variables typed directly into the console

The order of execution is also the override order. For example, console variables set in code override those set in configuration files (and `level.cfg` overrides `user.cfg`, and so on). Console variables set in flow graphs override any identical console variables set in code. Finally, console variables typed directly into the console override all the other console variable settings.

### Configuring Console Variables in the Console

You can specify values for console variables in the console to apply changes to your level.

**To configure console variables in the console**

- Do one of the following:
  - In the Console Variables window, search for the variable name, double-click the value, and then enter the value that you want.
  - In the Console command line, enter the console variable and its value, and then press Enter. For example, enter the `r_DisplayInfo=1` command to display debugging information in the viewport.
Configuring Console Variables in Configuration Files

You can specify values for console variables in the configuration files, such as the level configuration file (level.cfg).

**To configure console variables with a configuration file**

1. Navigate to the directory that has the configuration file. For example, if you want to configure the level.cfg file, navigate to the `lumberyard_version\dev\ProjectName\Levels\level_name` directory.
2. Use a text editor to edit the file or to create one.
3. Specify the console variable name and the value. For example: `r_DisplayInfo=1` shows debugging information in the viewport.
4. Save the file.

Console in Depth

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The console is a user interface system which handles console commands and console variables. It also outputs log messages and stores the input and output history.

Color coding

The game console supports color coding by using the color indices 0..9 with a leading $ character. The code is hidden in the text outputted on the console. Simple log messages through the ILog interface can be used to send text to the console.

```
This is normal $1one$2two$3three and so on
```

In the preceding example, one renders in red, two in green, and three (and the remaining text) in blue.

Console Variables

Console variables provide a convenient way to expose variables which can be modified easily by the user either by being entered in the console during runtime or by passing it as command-line argument before launching the application.

More information on how to use command-line arguments can be found in the Command Line Arguments article.

Console variables are commonly referred to as `CVar` in the code base.

Registering new console variables

For an integer or float based console variable, it is recommended to use the `IConsole::Register()` function to expose a C++ variable as a console variable.

To declare a new string console variable, use the `IConsole::RegisterString()` function.

Accessing console variables from C++

Console variables are exposed using the `ICVar` interface. To retrieve this interface, use the `IConsole::GetCVar()` function.
The most efficient way to read the console variable value is to access directly the C++ variable bound to the console variable proxy.

Adding New Console Commands

The console can easily be extended with new console commands. A new console command can be implemented in C++ as a static function which follows the `ConsoleCommandFunc` type. Arguments for this console command are passed using the `IConsoleCmdArgs` interface.

The following code shows the skeleton implementation of a console command:

```cpp
static void RequestLoadMod(IConsoleCmdArgs* pCmdArgs);

void RequestLoadMod(IConsoleCmdArgs* pCmdArgs)
{
    if (pCmdArgs->GetArgCount() == 2)
    {
        const char* pName = pCmdArgs->GetArg(1);
        // ...
    }
    else
    {
        CryLog("Error, correct syntax is: g_loadMod modname");
    }
}
```

The following code will register the command with the console system:

```cpp
IConsole* pConsole = gEnv->pSystem->GetIConsole();
pConsole->AddCommand("g_loadMod", RequestLoadMod);
```

Console Variable Groups

Console variable groups provide a convenient way to apply predefined settings to multiple console variables at once.

Console variables are commonly referred to as `CVarGroup` in the code base. Console variable groups can modify other console variables to build bigger hierarchies.

Warning
Cycles in the assignments are not detected and can cause crashes.

Registering a new variable group

To register a new variable group, add a new `.cfg` text file to the `GameSDK(config\CVarGroups` directory.

```cfg
sys_spec_Particles.cfg

[default]
; default of this CVarGroup
= 4
e_particles_lod=1
e_particles_max_emitter_draw_screen=64

[1]
e_particles_lod=0.75
e_particles_max_emitter_draw_screen=1

[2]
```

Version 1.28
214
This creates a new console variable group named `sys_spec_Particles` that behaves like an integer console variable. By default, this variable has the state 4 (set in the line following the comment in the example).

On changing the variable, the new state is applied. Console variables not specified in the `.cfg` file are not set. All console variables need to be part of the default section. An error message is output in case of violation of this rule.

If a console variable is not specified in a custom section, the value specified in the default section is applied.

**Console variable group documentation**

The documentation of the console variable group is generated automatically.

### `sys_spec_Particles`

Console variable group to apply settings to multiple variables

```plaintext
sys_spec_Particles [1/2/3/4/x]:
... e_particles_lod = 0.75/1/1/1/1
... e_particles_max_screen_fill = 16/32/64/128/128
... e_particles_object_collisions = 0/1/1/1/1
... e_water_ocean_soft_particles = 0/1/1/1/1
... r_UseSoftParticles = 0/1/1/1/1
```

Checking if a console variable group value represents the state of the variables it controls

**From the console**

In the console you can enter in the console variable group name and press tab. If the variable value is not represented, it will print the value of `RealState`.

```plaintext
sys_spec_Particles=2 [REQUIRE_NET_SYNC] RealState=3
sys_spec_Sound=1 [REQUIRE_NET_SYNC] RealState=CUSTOM
sys_spec_Texture=1 [REQUIRE_NET_SYNC]
```

By calling the console command `sys_RestoreSpec` you can check why the `sys_spec_` variables don't represent the right states.

**From C++ code**

From the code you can use the member function `GetRealIVal()` and compare its return value against the result of `GetIVal()` in `ICVar`.

**Deferred execution of command line console commands**

The commands that are passed via the command line by using the `+` prefix are stored in a separate list as opposed to the rest of the console commands.

This list allows the application to distribute the execution of those commands over several frames rather than executing everything at once.
Example

Consider the following example.

```bash
--- autotest.cfg --
hud_startPaused = "0"
wait_frames 100
screenshot autotestFrames
wait_seconds 5.0
screenshot autotestTime

-- console --
StarterGameLauncher.exe -devmode +map SinglePlayer +exec autotest +quit
```

In the example, the following operations were performed:

- Load the SinglePlayer map.
- Wait for 100 frames.
- Take a screenshot called autotestFrames.
- Wait for 5 seconds.
- Take a screenshot called autotestTime.
- Quit the application.

Details

Two categories of commands are defined: blocker and normal.

For each frame, the deferred command list is processed as a fifo. Elements of this list are consumed as long as normal commands are encountered.

When a blocker is consumed from the list and executed, the process is delayed until the next frame. For instance, commands like `map` and `screenshot` are blockers.

A console command (either command or variable) can be tagged as a blocker during its declaration using the `VF_BLOCKFRAME` flag.

The following synchronization commands are supported.

### Optional Title

<table>
<thead>
<tr>
<th>Command</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait_frames</td>
<td>num:</td>
<td>&lt;int&gt; Wait for <code>num</code> frames before the execution of the list is resumed.</td>
</tr>
<tr>
<td>wait_seconds</td>
<td>sec:</td>
<td>&lt;float&gt; Wait for <code>sec</code> seconds before the execution of the list is resumed.</td>
</tr>
</tbody>
</table>

Console Variable Tutorial

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This tutorial shows you how to modify existing and create console variables (CVARs). CVARs can be used to control many configurable behaviors in Lumberyard. You can also use them in your game.
Note
This brief tutorial is intended for programmers. Most of the content uses code.

Creating CVars

To create a console variable

1. In your code editor, open the Code\GameSDK\GameDll\GameCVars.h file, which declares all game-specific CVars.
2. Locate the SCVars struct. Inside the struct, declare a new variable, as in the following example.

```c
struct SCVars
{
    int g_tutorialVar; //add this line

    //... pre-existing code ...
};
```

The variable you added will be used to store the current value of the variable. If you need to store fractional numbers, you can also add a variable of the type float.

Next, you will register the CVar with the game engine so that its value can be changed by using the console.

3. In the same Code\GameSDK\GameDll\GameCVars.cpp file, locate and use the InitCVars function.

```c
void SCVars::InitCVars(IConsole *pConsole)
{
    m_releaseConstants.Init( pConsole );

    REGISTER_CVAR(g_tutorialVar, 42, VF_NULL, "This CVar was added using the tutorial on CVars"); //add this line

    //... pre-existing code ...
}
```

Specify a default value and help text for the variable. You can initialize the variable with any value that is valid for the type with which the variable was declared in the header file. The preceeding example specifies 42 as the default value and some help text that will be shown to users.

4. When your game unloads, be sure to un-register the variable. In the Code\GameSDK\GameDll\GameCVars.cpp file, locate and use the ReleaseCVars function, as shown in the following example.

```c
void SCVars::ReleaseCVars()
{
    IConsole* pConsole = gEnv->pConsole;

    pConsole->UnregisterVariable("g_tutorialVar", true); //add this line

    //... pre-existing code ...
}
```

6. After you finish making changes, don't forget to compile your code.

Using the CVar

You can now change the value of the CVar that you created by using code, the console, and .cfg files.

From code

Version 1.28

217
To access the value of the variable in your game code, use the `g_pGameCVars` pointer, as shown in the following example.

```cpp
int myTutorialVar = g_pGameCVars->g_tutorialVar;
```

**From the console**

To change the value of the cvar from the console, use the syntax `cvar_name=cvar_value`. The following example sets the value of the `g_tutorialVar` console variable to 1337.

```plaintext
g_tutorialVar = 1337
```

**From .cfg files**

It's also possible to change the default CVar value from one of the `.cfg` files. Whenever a CVar is assigned a value, its previous value is discarded. Therefore, the last assignment is the one that is current.

The following list shows the order of initialization for console variables.

1. The value specified in the `GameCVars.cpp` file when `REGISTER_CVAR` is used. (A change here requires compiling.)
2. The value specified in the `system.cfg` file.
3. The value specified in the user's `user.cfg` file.
4. Any value assigned at game runtime.

**Tip**

To change the default value of an existing CVar without having to compile, add a line to the `system.cfg` file to override the default.

---

**Automating the Lumberyard Editor with the Python Editor Bindings gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Some tasks in the Lumberyard Editor are tedious or could easily be automated, and to support that, Lumberyard has support for scripting the editor through Python bindings to the underlying editor implementation. These bindings are enabled with the `PythonEditorBindings` gem, and interacted with through a Python 3 library embedded within the editor. You can access a Python REPL through an in-editor console, or launch the editor with an argument that loads and runs a Python script on boot.

**Enable editor automation**

Editor automation is enabled by selecting the `PythonEditorBindings` gem for your project, and then rebuilding the editor. No specific configuration (debug, profile, release) is required to enable the Python bindings. Because the bindings are enabled through a gem that you select for your project, you’ll need to make sure that this gem is enabled for all projects that you intend to use automation with.

**Important**

In order to use the editor bindings, your Lumberyard installation needs to be configured to be able to build the editor and tools. See Running Lumberyard Setup Assistant (p. 16) for details.
Use editor automation

The easiest way to get started with editor automation is to use the REPL that's available from within the Lumberyard Editor and try out some commands. Open this REPL by selecting **Tools > Other > Python Console**. The Python console opens in a new editor view, which gives you access to a console that displays output from Python, the REPL input, and a full reference of available commands. To get access to the reference, select the ? icon in the lower-right corner of the console.

You can also access a set of available scripts, including some samples for common tasks in the editor, by selecting **Tools > Other > Python Scripts**. These scripts are stored in a directory depending on their scope. Scripts only for your project are stored in the `lumberyard_install\dev\Editor\Scripts` directory, and scripts meant to be used along with a gem are stored at `lumberyard_install\dev\Gems\gem\Gemname\Editor\Scripts`.

Editor automation is driven primarily through the event bus (EBus) system. Before working with the editor bindings, you should become familiar with the basics of EBus from Working with the Event Bus (EBus) system (p. 1851). To learn about some of the specific busses used by the editor automation system, take a look at the Python Editor Bindings gem examples (p. 219).

Python Editor Bindings gem examples

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Python Editor Bindings are driven by an API that connects through to the Editor's C++ implementation, using the Lumberyard event bus (Ebus) to send messages between scripts and the editor. This reference covers the use of the editor bindings API to perform tasks like interacting with components, entities, and and properties.

Contents

- Level management (p. 219)
- Editor timing (p. 220)
- Entities (p. 220)
  - Entity IDs (p. 220)
  - Entity operations and Ebus interfaces (p. 220)
  - Entity search (p. 221)
  - Entity notifications (p. 222)
- Component management (p. 223)
  - Component type events (p. 223)
  - Component usage events (p. 224)
  - Component control events (p. 225)
- Component property events (p. 226)
- Editing properties (p. 227)
  - Property containers (p. 228)
- Asset management (p. 229)

Level management

Use these functions to load, create, and save levels. In order to use other editor binding APIs, a level needs to be loaded in the editor.
Examples

# opens a level with a user prompt
azlmbr.legacy.general.open_level(strLevelName)

# opens a level without prompting the user (better for automation)
azlmbr.legacy.general.open_level_no_prompt(strLevelName)

# creates a level with the parameters of 'levelName', 'resolution', 'unitSize' and 'bUseTerrain'
azlmbr.legacy.general.create_level(levelName, resolution, unitSize, bUseTerrain)

# same as create_level() but no prompts
azlmbr.legacy.general.create_level_no_prompt(levelName, resolution, unitSize, bUseTerrain)

# saves the current level
azlmbr.legacy.general.save_level()

Editor timing

Occasionally a script will need to introduce a delay in actions to be performed in the editor while another action completes, such as loading a level. Rather than use the built-in Python delay methods, use these editor binding APIs.

# enables/disables idle processing for the Editor
azlmbr.legacy.general.idle_enable(boolValue)

# Returns whether or not idle processing is enabled for the Editor
azlmbr.legacy.general.is_idle_enabled()

# waits idling for a given seconds
azlmbr.legacy.general.idle_wait(floatSeconds)

Entities

The API allows you to add and remove entities to the root entity of a level, retrieve and compare entity IDs, and search for entities.

Entity IDs

The azlmbr.entity.EntityId class is used to refer to entity instances, properties, and the entity tree.

# returns True if the entity ID is valid
entityId.IsValid()

# returns string representation of an entity ID
entityId.ToString()

# returns True if both entity IDs
entityId.Equal(otherEntityId)

Entity operations and Ebus interfaces

There are three main EBus interfaces used to manage Editor entities:

- azlmbr.editor.ToolsApplicationRequestBus: Used to create and delete Editor entities
- azlmbr.editor.EditorEntityInfoRequestBus: Used to access Entity values
- azlmbr.editor.EditorEntityAPIBus: Used to mutate Entity values
Example usage:

```python
# Create a new Entity at the root level
rootEntityId = azlmbr.editor.ToolsApplicationRequestBus(azlmbr.bus.Broadcast, 'CreateNewEntity', EntityId())

# Create a new Entity parented to the parent Entity
childEntityId = azlmbr.editor.ToolsApplicationRequestBus(azlmbr.bus.Broadcast, 'CreateNewEntity', rootEntityId)

# Delete the entity
azlmbr.editor.ToolsApplicationRequestBus(azlmbr.bus.Broadcast, 'DeleteEntityById', childEntityId)

# Delete the root Entity we created and all its children
azlmbr.editor.ToolsApplicationRequestBus(azlmbr.bus.Broadcast, 'DeleteEntityAndAllDescendants', rootEntityId)

# Get current name
name = azlmbr.editor.EditorEntityInfoRequestBus(azlmbr.bus.Event, 'GetName', entityId);

# Set a new name
azlmbr.editor.EditorEntityAPIBus(azlmbr.bus.Event, 'SetName', entityId, "MyName")

# get the parent ID of this EntityID
getId = azlmbr.editor.EditorEntityInfoRequestBus(azlmbr.bus.Event, 'GetParent', childId);
```

**Entity search**

The entity search API is based around setting up filters using azlmbr.entity.SearchFilter to set up the search parameters, and then conduct the search over the Ebus represented by azlmbr.entity.SearchBus.

**azlmbr.entity.SearchFilter usage:**

```python
searchFilter = azlmbr.entity.SearchFilter()
searchFilter.names = [] # List of names (matches if any match); can contain wildcards in the name.
searchFilter.names_case_sensitive = False # Determines if the name matching should be case sensitive.
searchFilter.components = {} # Dictionary keyed on component type IDs (matches if anymatch).
searchFilter.components_match_all = False # Determines if the filter should match all component type ids (AND).
ssearchFilter.roots = [] # Specifies the entity IDs that act as roots of the search
searchFilter.names_are_root_based = False # Determines if the names are relative to the root or should be searched in children too.
```

**azlmbr.entity.SearchBus usage:**

```python
# The SearchBus interface
busType = azlmbr.bus.Broadcast

# Iterates through all entities in the current level, and returns a list of the ones that match the conditions
entityIdList = azlmbr.entity.SearchBus(busType, 'SearchEntities', searchFilter)

# Returns a list of all editor entities at the root level in the current level
entityIdList = azlmbr.entity.SearchBus(busType, 'GetRootEditorEntities', searchFilter)
```

Searching with wildcards:
Entities are addressable by a 'name path' using strings separated by the pipe character | such as root name|my entity|my child for a name path. Entity search also supports the use of ? and * wildcards.

**Example usage:**

```python
import azlmbr.bus as bus
import azlmbr.entity as entity

searchFilter = entity.SearchFilter()
searchFilter.names = ['TestName']

# Search by name
entityIdList = entity.SearchBus(bus.Broadcast, 'SearchEntities', searchFilter)

# Search by name path (DAG)
searchFilter = entity.SearchFilter()
searchFilter.names = ['TestParent|TestChild']
entityIdList = entity.SearchBus(bus.Broadcast, 'SearchEntities', searchFilter)

# Search using wildcard
searchFilter = azlmbr.entity.SearchFilter()
searchFilter.names = ['Test*|?estChild']
entityIdList = entity.SearchBus(bus.Broadcast, 'SearchEntities', searchFilter)

# Filter with roots
searchFilter = entity.SearchFilter()
searchFilter.names = ['TestChild']
searchFilter.roots = [rootId]
searchFilter.names_are_root_based = False # default
entityIdList = entity.SearchBus(bus.Broadcast, 'SearchEntities', searchFilter)

# Filter with roots using the names, only get the kids realtive from the root nodes
searchFilter = entity.SearchFilter()
searchFilter.names = ['TestParent|TestChild']
searchFilter.roots = [rootId]
searchFilter.names_are_root_based = True # search from roots for these names
entityIdList = entity.SearchBus(bus.Broadcast, 'SearchEntities', searchFilter)
```

**Entity notifications**

You can capture editor entity events using the `EditorEntityContextNotificationBus` handler. Callbacks can be assigned to entity management event names: `OnEditorEntityCreated` and `OnEditorEntityDeleted` where the callback will be called with a tuple of data coming from the events.

```python
# The events
"OnEditorEntityCreated" # returns when an entity is created in the Editor
"OnEditorEntityDeleted" # returns when an entity is destroyed in the Editor
```

**Example usage:**

```python
# assumes a level has been opened or created
import azlmbr.bus as bus
import azlmbr.editor as editor
from azlmbr.entity import EntityId
```
createdEntityIds = [] # to capture created entites

def onEditorEntityCreated(parameters):
    global createdEntityIds
    entityId = parameters[0]
    createdEntityIds.append(entityId)

def onEditorEntityDeleted(parameters):
    global createdEntityIds
    deletedEntityId = parameters[0]
    for entityId in createdEntityIds:
        if (entityId.Equal(deletedEntityId)):
            createdEntityIds.remove(entityId)
        break

# Listen for notifications when entities are created/deleted
handler = editor.EditorEntityContextNotificationBusHandler()
handler.connect() # connects to a Singleton bus handler
handler.add_callback('OnEditorEntityCreated', onEditorEntityCreated)
handler.add_callback('OnEditorEntityDeleted', onEditorEntityDeleted)

# Create new Editor Entity
editor.ToolsApplicationRequestBus(bus.Broadcast, 'CreateNewEntity', EntityId())

Component management

The component system is used to add and removes components to existing entities with the azlmbr.editor.EditorComponentAPIBus bus.

Note
Components are not active when in editing mode. They only become active when the game is being played within the editor.

Component type events

The API requires IDs to create, use, or control component instances. To get a component IDs, use the following Ebus events:

# azlmbr.editor.EditorComponentAPIBus Broadcast events

# Finds the component ids from their type names
# input: list of strings of type names
# output: (list of component type IDs)
'FindComponentTypeIds'

# Finds the component names from their type ids
# input: list of component type IDs
# output: (list of strings) of type names
'FindComponentTypeNameList'

Example usage:

import azlmbr.bus as bus
Component usage events

The API can add components to an existing entity, test for component existence, counts components by type, and enumerate the components on an entity.

Example usage:

```python
import azlmbr.bus as bus

# adding a Mesh component
meshComponentOutcome = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast,'AddComponentsOfType', entityID, [meshComponentTypeId])
if (meshComponentOutcome.IsSuccess()):
    print("Mesh component added to entity.")
```
```python
meshComponents = meshComponentOutcome.GetValue()
meshComponent = meshComponents[0]

# test for a mesh component exists on the entity
hasComponent = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'HasComponentOfType', entityId, meshComponentTypeId)
if (hasComponent):
    print("Entity has a Mesh component.")

# find the number of Mesh components on the entity
commentsCount = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'CountComponentsOfType', entityId, meshComponentTypeId)
if(commentsCount == 1):
    print("Entity has one Mesh component")

# returns the first mesh component ID, if any
meshSingleComponentOutcome = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'GetComponentOfType', entityId, meshComponentTypeId)
if (meshSingleComponentOutcome.IsSuccess()):
    print("GetComponentOfType mesh works.")
    firstMeshComponentId = meshSingleComponentOutcome.GetValue()

# returns a list of component IDs for a component type
meshMultipleComponentOutcome = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'GetComponentsOfType', entityId, meshComponentTypeId)
if (meshMultipleComponentOutcome.IsSuccess()):
    print("GetComponentsOfType mesh works.")
    firstMeshComponentId = meshMultipleComponentOutcome.GetValue()[0]

Component control events

The API offers events to validate, enable or disable, and remove components.

```
'RemoveComponents'

Example usage:

```python
import azlmbr.bus as bus

# test a component is valid
isValid = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'IsValid', meshComponent)
if (isValid is True):
    print("Mesh component is valid.")

# test if the component is enabled
isEnabled = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'IsComponentEnabled', meshComponent)
if (isEnabled is True):
    print("Mesh component is enabled.")

# enable this mesh component
isEnabled = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'EnableComponents', [meshComponent])
if (isEnabled is True):
    print("Mesh component set to enabled.")

# disable this mesh component
didDisable = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'DisableComponents', [meshComponent])
if (didDisable is True):
    print("Mesh component set to disabled.")

# remove only this mesh component
didRemove = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'RemoveComponents', [meshComponent])
if (didRemove is True):
    print("Mesh component has been removed.")
```

Component property events

Component properties can be accessed and modified using a string that indicates a direct path to a property value. The pipe character | is used as the separator between the property path elements.

The azlmbr.editor.EditorComponentAPIBus bus is used to access or modify component property values.

```python
# azlmbr.editor.EditorComponentAPIBus Broadcast events

# Get value of a property on a component
# input: component ID
# input: property path
# output: (Outcome<object>) the current value of the property
'GetComponentProperty'

# Set value of a property on a component
# input: component ID
# input: property path
# input: object value
# output: (Outcome<object>) the new value of the property
'SetComponentProperty'

# Get a full list of properties in a component
# input: component ID
# output: (list of strings) property paths
'BuildComponentPropertyList'
```
Example usage:

```python
import azlmbr.bus as bus

# Get current value of the mesh asset property of the MeshComponentRenderNode
propertyPath = "MeshComponentRenderNode|Mesh asset"
valueOutcome = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'GetComponentProperty', componentId, propertyPath)

if (valueOutcome.IsSuccess()):
    meshAssetId = valueOutcome.GetValue()
    print ('Old mesh asset is {}'.format(meshAssetId))

# Set the mesh asset
outcome = None
if (meshAssetId is not None):
    outcome = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'SetComponentProperty', componentId, propertyPath, meshAssetId)

if(outcome.IsSuccess()):
    result = outcome.GetValue()
    print ('New mesh asset is {}'.format(result))

# Read the properties of this MeshComponentRenderNode
propertyPaths = azlmbr.editor.EditorComponentAPIBus(bus.Broadcast, 'BuildComponentPropertyList', componentId)
for path in propertyPaths:
    print ('ComponentId path has {}'.format(path))
```

Editing properties

To access this API a script needs access to a property tree editor instance. This object accesses the properties on a component, in the style of the property editing view inside of the editor. Properties are accessed starting from the root of the component, and follow the chain of labels until a property value is encountered.

A common way to create a property tree editor instance is during content creation when a component is created via the EditorComponentAPIBus.AddComponentsOfType event.

```python
componentOutcome = editor.EditorComponentAPIBus(bus.Broadcast, 'AddComponentsOfType', entityId, typeIdsList)
if (!componentOutcome.IsSuccess()):
    raise Exception('FAILURE FATAL: AddComponentsOfType')

components = componentOutcome.GetValue()
pteObj = editor.EditorComponentAPIBus(bus.Broadcast, 'BuildComponentPropertyTreeEditor', components[0])
if(pteObj.IsSuccess()):
    pte = pteObj.GetValue()

azlmbr.property.PropertyTreeEditor API:

# type: azlmbr.property.PropertyTreeEditor
# - method: build_paths_list() -> string List
#     Get a complete list of all property paths in the tree.
# - method: build_paths_list_with_types() -> string List
#     Get a complete list of all property paths in the tree with (typename)s.
# - method: set_visible_enforcement() -> string List
```
Limits the properties using the visibility flags such as ShowChildrenOnly.
- method: has_attribute(str: path, str: attribute) -> bool
  Detects if a property has an attribute.
- method: get_value(str: path) -> Object
  Gets a property value.
- method: set_value(str: path, object: value)
  Sets a property value.
- method: compare_value(str: path, object: value) -> Boolean
  Compares a property value.
- method: is_container(str: path) -> Boolean
  True if property path points to a container.
- method: get_container_count(str: path) -> Outcome Integer
  Returns the size of the container.
- method: reset_container(str: path) -> Outcome Boolean
  Clears the items in a container.
- method: add_container_item(str: path, object key, object value) -> Outcome Boolean
  Add an item in a container.
- method: append_container_item(str: path, object value) -> Outcome Boolean
  Appends an item in an non-associative container.
- method: remove_container_item(str: path, object key) -> Outcome Boolean
  Removes a single item from a container.
- method: update_container_item(str: path, object key, object value) -> Outcome
  Updates an existing the item's value in a container.
- method: get_container_item(str: path, object: key) -> Outcome Object
  Retrieves an item value from a container.

Property containers

The Editor automaton API exposes a number of special methods to handle container component property types. If the property tree editor points to a component that has container properties these methods give access to the items in the container.

To determine if the property is a container type use the azlmbr.PropertyTreeEditor.is_container() method.

Example usage:

```python
# the path to the 'Extended Tags' property
tagListPropertyPath = 'm_template|Extended Tags'

# get current item count of the the container
outcome = pte.get_container_count(path)
if(outcome.IsSuccess()):
    count = outcome.GetValue()

# clear the container
outcome = pte.reset_container(path)
if(outcome.IsSuccess()):
    print('cleared item')

# if this is a Dictionary type make sure to have a valid key
key = 'tag_1'
value = pte.add_container_item(path, key, value)
if(outcome.IsSuccess()):
    print('added item')

# an update needs a key such as an index or a Dictionary key
value = 'tag_2'
outcome = pte.update_container_item(path, key, value)
if(outcome.IsSuccess()):
    print('updated item')
```
Examples

```python
print('updated an item')

# the 'append' can be used for properties that are Lists
value = 'tag_3'
ooutcome = pte.append_container_item(path, value)
if(outcome.IsSuccess()):
    print('appended an item')

# get an item using a key such as an index or a Dictionary key
key = 0
outcome = pte.get_container_item(path, key)
if(outcome.IsSuccess()):
    print('got the value {} from index 0'.format(outcome.GetValue()))

# remove an item using a key,
# even in List types give an index for the key
key = 0
outcome = pte.remove_container_item(path, key)
if(outcome.IsSuccess()):
    print('removed an item')
```

Asset management

The editor automation API exposes a few methods to manage assets via the `azlmbr.asset.AssetCatalogRequestBus` bus.

```python
import azlmbr.bus as bus
import azlmbr.math
emptyTypeId = azlmbr.math.Uuid()

# get the cube asset ID
bRegisterType = False
cubeAssetId = azlmbr.asset.AssetCatalogRequestBus(bus.Broadcast, 'GetAssetIdByPath',
        'objects/default/primitive_cube.cgf', emptyTypeId, bRegisterType)
print ('cube asset ID validity is {}'.format(cubeAssetId.IsValid()))

cubePath = azlmbr.asset.AssetCatalogRequestBus(bus.Broadcast, 'GetAssetPathById',
        cubeAssetId)
print ('cube asset path is {}'.format(cubePath))
```
Demo and Video Capture

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This section contains information about recording videos for benchmarking. Capturing audio and video is also discussed, using either the Perspective viewport in Lumberyard Editor or in game mode using the launcher.

Topics
- Capturing Video and Audio (p. 230)
- Recording Time Demos (p. 233)

Capturing Video and Audio

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This tutorial explains how to set up Lumberyard editor (or game) to capture video. Lumberyard outputs video as single frames. If required, it can also output stereo or 5.1 surround sound audio in .wav file format. You can edit the output with commonly available video editing software.

Preparation

Before you can start video and audio streams in preparation for capture, you must configure some settings that determine how the video will be captured. You configure these settings by using console commands. To save time, you can create configuration files that execute the necessary commands for you instead of entering the commands directly into the console. Example configuration files are presented later in this topic.

The next sections describe the settings and the console commands that configure them.

Video Settings

Frame Size and Resolution

The height and width of the captured frames in the editor is normally set to the exact view size of your rendered perspective window. To resize the view size, re-scale the perspective window, or right click in the top right of the perspective viewport where the frame size is displayed.

You can also capture higher than rendered images from Lumberyard Editor and Launcher.

The console variables that are now used in conjunction with Capture Frames are:

- r_CustomResHeight=N - Specifies the desired frame height in N pixels.
- r_CustomResWidth=M - Specifies the desired frame width in M pixels.
- r_CustomResMaxSize=P - Specifies the maximum resolution at which the engine will render the frames in P pixels.
- r_CustomResPreview=R - Specifies whether or how the preview is displayed in the viewport. Possible values for R are:
Frames Per Second

When deciding the number of frames per second to specify, keep in mind the following:

- NTSC standard video is approximately 30 frames per second, which is a good compromise between quality and file size.
- High quality video can have up to 60 frames per second, but the difference in quality of the increased number of frames is barely noticeable and can take up a lot of file space.
- Video at less than 24 FPS (a cinema standard) will not look smooth.

To specify a fixed frame rate, use the command:

```
t_fixedstep N
```

$N$ specifies the time step. Time step is calculated by using the formula

```
step = 1 second/<number of frames>
```

A table of common time step values follows.

<table>
<thead>
<tr>
<th>FPS</th>
<th>Time Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 (PAL)</td>
<td>0.04</td>
</tr>
<tr>
<td>30</td>
<td>0.0333333333</td>
</tr>
<tr>
<td>60</td>
<td>0.0166666667</td>
</tr>
</tbody>
</table>

Video Capture File Format

You can capture pictures in several different file formats. A good choice for average quality is the `.jpeg` file format. The `.tga` or `.bmp` file formats are better for higher quality, and `.hdr` for pictures that use high-dynamic-range imaging.

To specify the capture file format, use the console command

```
capture_file_format N
```

$N$ is jpg, bmp, tga or hdr.

Video Capture File Location

By default, recorded frames are stored in the directory $<root>\CaptureOutput$. To specify a custom directory, use the command:
**Lumberyard User Guide**

### Capturing Video and Audio

- **capture_folder** \(N\)

\(N\) is the name of the custom directory.

**Warning**

When you start a recording, the captured frames are placed in the currently specified directory and will overwrite existing files with the same name. To avoid losing work, create a directory for each recording, or move the existing files to another directory before you start.

### Starting and Ending the Video Recording

After you have specified the values mentioned in the previous sections, you can start the recording by using the command:

- **capture_frames** \(N\)

Setting \(N\) to 1 starts the recording, and setting \(N\) to 0 stops it.

### Audio Settings

Before you begin, decide if you require audio in stereo or in 5.1 surround format, and then change your audio settings accordingly in the Windows control panel.

### Deactivating the Sound System

After loading the level of your game that you want to capture, you must deactivate the sound system so that you can redirect the sound output to a file. To deactivate the sound system, use the command:

- \#Sound.DeactivateAudioDevice()

This redirects the sound output to a `.wav` file in the root directory of the game. The sound will not run in realtime, but be linked precisely to the time step that you set previously.

To write the sound capture, use the command:

- `s_OutputConfig N`

Setting \(N\) to 3 activates the non-realtime writing of sound to the `.wav` file. Setting \(N\) to 0 specifies auto-detection (the default).

### Reactivating the Sound System

To reset the sound system use the command:

- \#Sound.ActivateAudioDevice()

This creates a `.wav` file in the root directory of the game. The file will continue to be written to until you run the following combination of commands to deactivate the audio device:

- \#Sound.DeactivateAudioDevice()

- `s_OutputConfig 0`

- \#Sound.ActivateAudioDevice()
Tip
Although these commands reset the sound system, some sounds won’t start until they are correctly triggered again. This applies particularly to looped sounds. To get looped sounds to play, start the recording of video and sound first, and then enter any area that triggers the looped sounds that you want to record.

Configuration Files

Creating Configuration Files

- To ensure that multiple recordings use exactly the same settings, create a configuration file that you can use for each of them. This will ensure that all of your captured files have the same format.

An example configuration file:

```
sys_spec = 4
Fixed_time_step 0.0333333333
Capture_file_format jpg
Capture_folder myrecording
r_width 1280
r_height 800
```

The command `sys_spec = 4` sets the game graphic settings to "very high" to generate the best appearance.

- To speed up the process of starting and stopping the recording, you can create two configuration files: one to start the video, and one to stop it.

- To start recording, use a config file like the following:

```
#Sound.DeactivateAudioDevice()
s_OutputConfig 3
#Sound.ActivateAudioDevice()
Capture_frames 1
```

- To stop recording, use a config file like the following:

```
Capture_frames 0
#Sound.DeactivateAudioDevice()
s_OutputConfig 0
#Sound.ActivateAudioDevice()
```

Executing the Config Files

To run the config file, open the console and enter the following command:

```
Exec N
```

`N` is the name of the config file.

Recording Time Demos

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Overview

Lumberyard Editor can record and play back player input and camera movement.

**Note**
Recording of some player actions such as vehicle movement are not supported.

To use the feature, you must start game mode in Lumberyard Editor and then record in it. To start game mode, press `Ctrl+G` after a level has fully loaded, or load the level in pure-game mode.

Output like the following appears both in the console and in the `timedemo.log` file in the directory corresponding to the level used:

```
TimeDemo Run 131 Finished.
Play Time: 3.96s, Average FPS: 50.48
Min FPS: 0.63 at frame 117, Max FPS: 69.84 at frame 189
Average Tri/Sec: 14037316, Tri/Frame: 278071
Recorded/Played Tris ratio: 0.99
```

Recording Controls

<table>
<thead>
<tr>
<th>Command</th>
<th>Keystroke</th>
<th>Console Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Recording</td>
<td><code>Ctrl + PrintScreen</code></td>
<td>record</td>
</tr>
<tr>
<td>End Recording</td>
<td><code>Ctrl + Break</code></td>
<td>stoprecording</td>
</tr>
<tr>
<td>Start Playback</td>
<td><code>Shift + PrintScreen</code></td>
<td>demo</td>
</tr>
<tr>
<td>Stop Playback</td>
<td><code>Ctrl + Break</code></td>
<td>stopdemo</td>
</tr>
</tbody>
</table>

Related Console Variables

- `stopdemo` – Stops playing a time demo.
- `demo demoname` – Plays the time demo from the specified file.
- `demo_fixed_timestep` – Specifies the number of updates per second.
- `demo_panoramic` – Uses a panoramic view when playing back the demo.
- `demo_restart_level N` – Repeats the level after each loop. Possible values for `N`: 0 = Off; 1 = use quicksave on first playback; 2 = load level start.
- `demo_ai` – Enables or disables AI during the demo.
- `demo_savestats` – Saves level stats at the end of the loop.
- `demo_max_frames` – Specifies the maximum number of frames to save.
- `demo_screenshot_frame N` – Makes a screenshot of the specified frame during demo playback. If a negative value for `N` is supplied, takes a screenshot every `N` frame.
- `demo_quit` – Quits the game after the demo run is finished.
- `demo_noinfo` – Disables the information display during the demo playback.
- `demo_scroll_pause` – Enables the use of the ScrollLock key to pause demo play and record.
- `demo_num_runs` – Specifies the number of times to loop the demo.
- `demo_profile` – Enables demo profiling.
- `demo_file` – Specifies the time demo file name.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can customize your workspace by adjusting how the windows and tools are docked, customizing which toolbars and menus display, and updating the global editor settings.

**Note**

To change the size setting for the toolbar icon, open the `Editor.cfg` file and enter a value for the `ed_toolbarIconSize` parameter. By default, the toolbar icon size is set to 0 (32 pixels).

**Topics**

- Docking Windows and Toolbars (p. 235)
- Customizing Toolbars and Menus (p. 237)
- Changing Preferences (p. 238)

### Docking Windows and Toolbars

When you drag a window or toolbar over an interface element or the edges of the editor, docking targets appear to show you where you can dock them. These targets appear for the top, bottom, left, and right quadrants of the pane. You can dock windows relative to any open pane, whether it is already docked, floating as a tab, or split in a column or row.

**To split a row or column**

To split a row or column, drop the window or toolbar on a docking target.

**To dock a window or toolbar as a tab**
To dock a window or toolbar as a tab, drop it on the docking target in the middle of a pane.

To dock a window or toolbar to the editor window, drop it on the outer docking target. This creates a new column next to the existing column.

To undock a window or toolbar, drag the title bar and move the selection window away. Avoid the docking targets to prevent from accidentally redocking the window. To help prevent accidental docking,
a brief delay occurs before a docking target becomes active. You can also undock a window by right-clicking the title bar and choosing **Undock**.

To prevent a window from docking

To prevent a window from docking, press and hold **Ctrl** while moving the window.

**To snap windows**

To snap a window in place, move the window close to a stationary window. Snapping works on the top, bottom, left, and right borders of the pane.

**Customizing Toolbars and Menus**

You can also give toolbars and menus your own personal touch.

**To customize preset toolbars or create custom toolbars and menus**

1. Right-click the top toolbar and choose **Customize**.
2. In the **Customize** window, for **Toolbars**, create, rename, and delete any custom toolbars and menus, or reset them to the default settings.
Changing Preferences

You can change the default settings to customize the look and functionality of the editor.

To customize the look and features of Lumberyard Editor

- In Lumberyard Editor, choose **Edit, Editor Settings, Global Preferences**.
General Settings

You can change the general Lumberyard Editor settings and file settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Geometry Preview Panel</td>
<td>Displays a preview window for the selected object.</td>
</tr>
<tr>
<td>Hide objects by config spec</td>
<td>Hides objects as determined by the minimal specifications and configuration specifications.</td>
</tr>
<tr>
<td>Enable Source Control</td>
<td>Enables Perforce version control.</td>
</tr>
<tr>
<td>External Layers: Save only Modified</td>
<td>Saves only the modified external layers.</td>
</tr>
<tr>
<td>Freeze Read-only external layer on Load</td>
<td>Freezes the read-only external layers when loading the level.</td>
</tr>
<tr>
<td>Frozen layers are selectable</td>
<td>Allows objects in frozen layers to be selected.</td>
</tr>
<tr>
<td>Console Background</td>
<td>Changes the background color for the console.</td>
</tr>
<tr>
<td>Auto-load last level at startup</td>
<td>Loads the level that was last loaded.</td>
</tr>
</tbody>
</table>
## Changing Preferences

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Time in Console</td>
<td>Displays the time in the console window.</td>
</tr>
<tr>
<td>Toolbar Icon Size</td>
<td>Adjusts the toolbar icon size. Default: 16 (32 pixels).</td>
</tr>
<tr>
<td>Stylus Mode</td>
<td>Enables stylus mode for tablets and other pointing devices.</td>
</tr>
<tr>
<td>Enable UI 2.0 (EXPERIMENTAL)</td>
<td>Enables the updated user interface (UI) in Lumberyard Editor.</td>
</tr>
<tr>
<td>Enable Scene Inspector (EXPERIMENTAL)</td>
<td>Enables the option to inspect scenes in files such as .fbx files.</td>
</tr>
<tr>
<td>Restore Viewport Camera on Game Mode Exit</td>
<td>Returns the camera to the original transform when you exit gameplay mode.</td>
</tr>
<tr>
<td>Enable Legacy UI (DEPRECATED)</td>
<td>Enable the legacy user interface (UI) in Lumberyard Editor.</td>
</tr>
<tr>
<td>Enable New Viewport Interaction Model (EXPERIMENTAL)</td>
<td>Replaces the old interaction model in the viewport.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Working with the Viewport Interaction Model (p. 494).</td>
</tr>
</tbody>
</table>

### Messaging

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Welcome to Lumberyard at startup</td>
<td>Displays the Welcome to Lumberyard dialog box at startup.</td>
</tr>
<tr>
<td>Show Error: Circular dependency</td>
<td>Shows an error message when adding a slice instance to a target would create cyclic asset dependency. All other overrides to slices will be saved.</td>
</tr>
</tbody>
</table>

### Undo

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo Levels</td>
<td>Specifies the maximum number of times you can undo a level. Default: 50</td>
</tr>
<tr>
<td>Undo Slice Override Saves</td>
<td>Allows you to undo override saves to slices.</td>
</tr>
</tbody>
</table>

### Selection

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep selection range</td>
<td></td>
</tr>
<tr>
<td>Stick duplicate to cursor</td>
<td></td>
</tr>
</tbody>
</table>
### Vertex Snapping

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Cube Size</td>
<td>Adjusts the vertex cube size.</td>
</tr>
<tr>
<td>Render Penetrated Boundboxes</td>
<td>Renders penetrated bound boxes.</td>
</tr>
</tbody>
</table>

### Metrics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Metrics Tracking</td>
<td>Enables metrics tracking.</td>
</tr>
</tbody>
</table>

### Slices

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Slices Dynamic By Default</td>
<td>When you create a slice, the slice is set to dynamic.</td>
</tr>
</tbody>
</table>

### Files

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append numeric value to slices</td>
<td>Automatically adds a number to the slice file name. For example, if you create a slice from an entity named \textit{NewSlice}, the slice will be named \textit{NewSlice001.slice}. For more information, see Creating a Slice (p. 516).</td>
</tr>
<tr>
<td>Backup on Save</td>
<td>Creates a backup file when you save.</td>
</tr>
<tr>
<td>Maximum Save Backups</td>
<td>Specifies the maximum number of saved backups.</td>
</tr>
<tr>
<td>Standard Temporary Directory</td>
<td>Specifies the default temporary directory to use. Default = \texttt{[root]\Temp}</td>
</tr>
<tr>
<td>Auto Save Camera Tag Points</td>
<td>Saves the modified camera tag points.</td>
</tr>
<tr>
<td>Scripts Editor</td>
<td>Specifies the text editor to use for scripts.</td>
</tr>
<tr>
<td>Shaders Editor</td>
<td>Specifies the text editor to use for shaders.</td>
</tr>
<tr>
<td>BSpace Editor</td>
<td>Specifies the text editor to use for blend spaces.</td>
</tr>
<tr>
<td>Texture Editor</td>
<td>Specifies the program to use for textures.</td>
</tr>
<tr>
<td>Animation Editor</td>
<td>Specifies the program to use for animations.</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables auto backup.</td>
</tr>
<tr>
<td>Time Interval</td>
<td>Specifies the frequency of auto backup (in minutes).</td>
</tr>
<tr>
<td>Maximum Backups</td>
<td>Specifies the maximum number of auto backups.</td>
</tr>
<tr>
<td>Remind Time</td>
<td>Specifies the frequency of auto backup reminders (in minutes).</td>
</tr>
</tbody>
</table>
**Viewport**

You can change the default settings for the viewport.

**General**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronize 2D Viewports</strong></td>
<td>Enables synchronization of 2D viewports to move and correspond with each other.</td>
</tr>
<tr>
<td><strong>Perspective View FOV</strong></td>
<td>Specifies the field of vision for the viewport.</td>
</tr>
<tr>
<td><strong>Perspective View Aspect Ratio</strong></td>
<td>Specifies the length of the aspect ratio for the viewport, where height = 1.</td>
</tr>
<tr>
<td><strong>Enable Right-Click Context Menu</strong></td>
<td>Enables the context menu that appears when you right-click in the viewport.</td>
</tr>
<tr>
<td><strong>Show 4:3 Aspect Ratio Frame</strong></td>
<td>Displays a 4:3 aspect ratio frame to show what is visible in game mode.</td>
</tr>
<tr>
<td><strong>Highlight Selected Geometry</strong></td>
<td>Highlights the selected geometry.</td>
</tr>
<tr>
<td><strong>Highlight Selected Vegetation</strong></td>
<td>Highlights the selected vegetation.</td>
</tr>
<tr>
<td><strong>Highlight Geometry on Mouse Over</strong></td>
<td>Highlights geometry on hover over.</td>
</tr>
<tr>
<td><strong>Hide Cursor when Captured</strong></td>
<td>Shows or hides the mouse pointer in the viewport.</td>
</tr>
<tr>
<td><strong>Drag Square Size</strong></td>
<td>Specifies the size of the drag square to prevent from accidentally moving objects when selecting.</td>
</tr>
<tr>
<td><strong>Display Object Links</strong></td>
<td>Displays entity links in the viewport.</td>
</tr>
<tr>
<td><strong>Display Animation Tracks</strong></td>
<td>Displays the animation path for any objects in the Track View. One line = one frame.</td>
</tr>
<tr>
<td><strong>Always Show Radii</strong></td>
<td>Displays the area of effect (radius) for certain entities.</td>
</tr>
<tr>
<td><strong>Always ShowPrefab Bounds</strong></td>
<td>Displays the prefab boundary helpers.</td>
</tr>
<tr>
<td><strong>Always ShowPrefab Objects</strong></td>
<td>Displays the prefab object helpers.</td>
</tr>
<tr>
<td><strong>Show Bounding Boxes</strong></td>
<td>Displays a bounding box around each object.</td>
</tr>
<tr>
<td><strong>Always Draw Entity Labels</strong></td>
<td>Displays entity names.</td>
</tr>
<tr>
<td><strong>Always ShowTrigger Bounds</strong></td>
<td>Displays the trigger boundary helpers.</td>
</tr>
<tr>
<td><strong>Show Object Icons</strong></td>
<td>Displays object icons.</td>
</tr>
<tr>
<td><strong>Scale Object Icons with Distance</strong></td>
<td>Scales object icons relative to distance.</td>
</tr>
<tr>
<td><strong>Show Helpers of Frozen Objects</strong></td>
<td>Displays the frozen object helper icons.</td>
</tr>
<tr>
<td><strong>Fill Selected Shapes</strong></td>
<td>Highlights the inside area of a selected shape.</td>
</tr>
<tr>
<td><strong>Show Snapping Grid Guide</strong></td>
<td>Displays the grid in the viewport.</td>
</tr>
</tbody>
</table>
# Changing Preferences

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Dimension Figures</td>
<td>Displays the measurement dimensions of selected assets; you must enable helpers.</td>
</tr>
<tr>
<td>Swap X/Y Axis</td>
<td>Reverses the x-axis and y-axis.</td>
</tr>
<tr>
<td>Map Texture Resolution</td>
<td>Specifies the resolution for the displayed map.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Displays object names.</td>
</tr>
<tr>
<td>Distance</td>
<td>Specifies the visibility distance for text labels.</td>
</tr>
<tr>
<td>Prefab Bounding Box</td>
<td>Specifies the color for the prefab bounding box.</td>
</tr>
<tr>
<td>Group Bounding Box</td>
<td>Specifies the color for the group bounding box.</td>
</tr>
<tr>
<td>Entity Bounding Box</td>
<td>Specifies the color for the entity bounding box.</td>
</tr>
<tr>
<td>Bounding Box Highlight Alpha</td>
<td>Specifies the amount of highlight alpha to add to the bounding box.</td>
</tr>
<tr>
<td>Geometry Color</td>
<td>Specifies the geometry color.</td>
</tr>
<tr>
<td>Solid Brush Geometry Color</td>
<td>Specifies the color of the solid brush geometry.</td>
</tr>
<tr>
<td>Geometry Highlight Alpha</td>
<td>Specifies the amount of highlight alpha to add to the geometry.</td>
</tr>
<tr>
<td>Child Geometry Highlight Alpha</td>
<td>Specifies the amount of highlight alpha to add to the child geometry.</td>
</tr>
</tbody>
</table>

## Movement

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera Movement Speed</td>
<td>Specifies the speed of all movements in the viewport.</td>
</tr>
<tr>
<td>Camera Rotation Speed</td>
<td>Specifies the speed of movement while you control the viewport camera.</td>
</tr>
<tr>
<td>Fast Movement Scale</td>
<td>Specifies the multiplier for the camera speed; for example, a value of two doubles the movement speed of the camera.</td>
</tr>
<tr>
<td>Wheel Zoom Speed</td>
<td>Specifies the speed of the camera zoom when using the mouse wheel.</td>
</tr>
<tr>
<td>Invert Y Axis</td>
<td>Inverts the direction that the camera moves on the y-axis when holding the right button on the mouse and moving the mouse up or down.</td>
</tr>
<tr>
<td>Invert Pan</td>
<td>Inverts the direction that the camera moves when holding the middle button on the mouse and moving the mouse left or right.</td>
</tr>
</tbody>
</table>

## Gizmos

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Specifies the size of the xyz-axes gizmo.</td>
</tr>
<tr>
<td>Text Labels</td>
<td>Displays the xyz-axes labels.</td>
</tr>
</tbody>
</table>
Parameter | Description
--- | ---
Max Count | Specifies the maximum number of xyz-axes gizmos that can display onscreen at one time.
Helpers Scale | Specifies the size of onscreen helpers, including AIAnchors, Tagpoints, and CoverSurfaces.
Tagpoint Scale Multiplier | Specifies the scale of the tagpoint helper sphere and the base helper scale value.
Ruler Sphere Scale | Specifies the scale of the locator sphere size when using the Ruler tool.
Ruler Sphere Transparency | Specifies the transparency level of the locator sphere when using the Ruler tool.

Debug
Parameter | Description
--- | ---
Show Mesh Statistics | Displays the level of detail information, such as tris and verts, for selectable objects.
Warning Icons Draw Distance | Specifies the distance to which to display warning icons in the viewport.
Show Scale Warnings | Displays an icon and warning text for objects that have been scaled.
Show Rotation Warnings | Displays an icon and warning text for objects that have been rotated.

Experimental Features
You can change the default settings for experimental features such as total illumination.

Lighting Settings
Parameter | Description
--- | ---
Total Illumination | Enables the total illumination lighting feature

---

Restoring the Default Layout for Lumberyard Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you customized your workspace, you can reset Lumberyard Editor to use the default layout. This restores your editors, tools, and windows to the default view that you see when you open Lumberyard Editor for the first time. You can restore the default layout at any time.
Contents

- Restoring the Component Entity Layout (p. 245)
- Restoring the Legacy Layout (p. 245)

Restoring the Component Entity Layout

The component entity layout is the default view for Lumberyard Editor. If you customize your settings and tools, you can restore the default layout for Lumberyard Editor.

To use the component entity layout (default)

- In Lumberyard Editor, do one of the following:
  - Click View, Layouts, Component Entity Layout.
  - Click View, Layouts, Restore Default Layout.

The following example shows the component entity layout in Lumberyard Editor:

Note

- Resetting Lumberyard Editor to the default layout does not reset the layout for individual tools, such as the Script Canvas editor or the Particle Editor.
- You can enable the CryEntity Removal gem to disable all legacy features from Lumberyard Editor. Legacy features will eventually be removed. If you enable this gem, the legacy layout view is removed from Lumberyard Editor. For more information, see CryEntity Removal Gem (p. 1123).

Restoring the Legacy Layout

If you don't enable the CryEntity Removal gem, you can switch back to the legacy layout.
To use the legacy layout

- In Lumberyard Editor, click View, Layouts, Legacy Layout.

The following example shows the legacy layout in Lumberyard Editor:

If you require more granular control to restore Lumberyard Editor settings, you can delete the relevant keys in the Windows registry.

**Important**
Exercise caution when editing the Windows registry. Not following the instructions carefully can result in a corrupt Windows installation.

**To edit the Windows registry**

1. On your Windows desktop, click Start and enter regedit in the search box.
2. In the Registry Editor, go to HKEY_CURRENT_USER\Software\Amazon\Lumberyard\Editor.
3. Right-click the `fancyWindowLayouts` and `mainWindowLayouts` folder icons and choose Delete. This deletes custom layout settings that you have for Lumberyard Editor.

The default settings are restored the next time you start Lumberyard Editor.
## Working with assets in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

## Working with assets in Lumberyard

Learn how to manage, process, and package the assets for your Lumberyard game. This documentation covers the Lumberyard Asset Browser tool, the asset pipeline, managing and organizing game assets, and packaging (bundling) game assets for delivery.

### In this section:

<table>
<thead>
<tr>
<th>Topic Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset Pipeline (p. 247)</td>
<td>Learn about the Lumberyard asset pipeline and asset builders. Asset builders convert your raw art, shaders, and other materials into game-ready formats that can be imported straight into the Lumberyard editor.</td>
</tr>
<tr>
<td>Asset Processor (p. 248)</td>
<td>Learn how the Lumberyard Asset Processor tool manages and stores asset dependencies.</td>
</tr>
<tr>
<td>Custom Asset Builders (p. 304)</td>
<td>Learn how to create custom asset builders that support new or non-standard formats.</td>
</tr>
<tr>
<td>Asset Browser (p. 296)</td>
<td>Learn how the Lumberyard Asset Browser tool is used to view and interact with your game project assets.</td>
</tr>
<tr>
<td>Asset Bundler (p. 334)</td>
<td>Learn how the Lumberyard Asset Bundler can be used to manage and package your game assets for delivery.</td>
</tr>
<tr>
<td>FBX Settings (p. 409)</td>
<td>Learn how to use FBX Settings to export meshes, actors, motions and PhysX meshes for Lumberyard.</td>
</tr>
<tr>
<td>Python Asset Builder (p. 448)</td>
<td>Learn how to create custom asset builders with Python.</td>
</tr>
</tbody>
</table>

## Working with the Asset Pipeline and asset files

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Asset Pipeline converts source art and other assets into OS-specific, game ready data. To prepare your game to ship, build all your game assets with the Asset Pipeline and package them with your game for your supported operating systems.

The Asset Processor (AP) is a service that runs in the background and monitors a configurable set of input directories for changes in files. When it detects changes, it uses configurable rules to determine its next action. The objective is to end up with game-ready versions of all assets for each OS and each game directory in a location called the asset cache. The asset cache is kept separate from your input directory and can be automatically rebuilt entirely from your source assets by the Asset Processor.
Using Asset Processor

Note
The asset cache should not be added to your source control.

The Asset Processor detects changes in the directories that contain input assets, with the game directory being the highest priority. Therefore, if you put assets in the game directory, those assets override assets with the same path in Lumberyard or other directories with lower priority.

Each output directory in the asset cache represents a full image of all files (except for executables and related files) needed to run the game. The Asset Processor curates the directory to keep it up to date, ensuring that new files are ready to use in the game and Lumberyard Editor as soon as possible. Game runtimes load assets only from the asset cache and never directly from your input source folders.

Topics
- Using Asset Processor (p. 248)
- Configuring the Asset Pipeline (p. 269)
- Configuring Image Processing (p. 278)
- Live Reloading and VFS (p. 291)
- Compiling Shaders for Release Builds (p. 291)
- Shader Compiler Proxy (p. 292)
- Shader Cache and Generation (p. 293)
- Game Startup Sequence (p. 296)
- Asset Browser (p. 296)
- Programming the Lumberyard AZCore Runtime Asset System (p. 304)

Using Asset Processor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Asset Processor is a utility that runs in the background to detect changes to your asset files. When Asset Processor detects new or updated asset files, it launches the Resource Compiler (rc.exe for FBX and ABC files, AssetBuilder.exe for all other types), processes the assets, and then places them in the cache. Asset Processor then notifies all running game or tool instances that the assets are updated. The game can then reload the updated assets.
As part of Asset Processing, the Asset Processor generates and stores product and source dependencies. In this context, a dependency defines how a one product or source asset depends on another asset. A given asset may have 0 or more dependencies, and these dependencies are used by features such as the Asset Bundler (p. 334) in order to determine which assets must be included when you bundle your game for release.

**Topics**
- Modifying the Asset Processor Configuration File (p. 250)
- Using the Asset Processor Batch Program (p. 250)
- Asset Processor Interface (p. 250)
- Enabling Asset Processor's Faster Scanning Mode (p. 261)
- Importing Assets into Lumberyard (p. 263)
- Using Resource Compiler (p. 264)
- Debugging Asset Processor (p. 265)

Asset Processor enables games to run on other platforms without deploying assets to that platform. Instead, the assets are accessed from the asset cache on a connected Windows or macOS system. With Asset Processor, you can also run games that use someone else's assets.

By proxying requests through itself, Asset Processor communicates with an iOS or Android shader compiler server through a USB cable on iOS and Android.

In Windows, Asset Processor starts automatically if you run Lumberyard Editor with automatically maintained connections. Asset Processor also restarts automatically if you modify any of the data files that it needs to operate or if you retrieve a new version.

In macOS, you must manually start Asset Processor from a command line window. Asset Processor is located in the `lumberyard_version/dev/BinMac64` directory.

**Note**
Symbolic links are not supported when using Asset Processor in macOS. To ensure that Asset Processor works properly in macOS, follow these guidelines:

- Do not use a symbolic link for your cache directory when you store compiled assets in a central location.
- Do not store your source project assets in a symbolic link directory.
- Use a unique cache directory. Do not share the cache directory with a Windows system that is also running Asset Processor.

You can open the Asset Processor options from the notification area on the taskbar.
You don't need to close Asset Processor when you get the latest updates from source control. You can start Lumberyard Editor while Asset Processor is still processing your assets.

However, if you aren't using the game or Lumberyard Editor, you can exit Asset Processor by right-clicking its icon in the notification area on the Windows taskbar or the macOS menu bar.

Asset Processor can also serve files directly to running console games so that the assets aren't required to be present on the game device. This is called virtual file system (VFS) and is required for live reloading to work on those platforms. For more information, see Live Reloading and VFS (p. 291).

Modifying the Asset Processor Configuration File

Use the `AssetProcessorPlatformConfig.ini` configuration file (located in the `lumberyard_version/dev/` directory) to perform the following tasks:

- Add new file types for Asset Processor to feed to the Resource Compiler, copy into the cache, or update existing file type rules.
- Update the ignore list.
- Specify which platforms are currently enabled. The default value is the host platform that Asset Processor runs on. Asset Processor automatically builds assets for the host platform. For example, if Asset Processor is running on Windows, Asset Processor builds Windows assets even if `pc` is not enabled in the `.ini` file. If Asset Processor is running on macOS, Asset Processor builds macOS assets even if `osx_gl` is not enabled in the `.ini` file. To build assets for other platforms, update the `.ini` file and specify the platforms that you want.
- Add additional folders for Asset Processor to watch. For example, you can specify folders such as shared particle libraries and associated textures between projects.
- Specify which files trigger related files to be rebuilt. This is called metafile fingerprinting.

To add game-specific overrides, you can add a file named `AssetProcessorGamePlatformConfig.ini` to your game assets directory. This file is read after the root configuration file and can have additional game-specific settings for the ignore list, platforms, and file types.

For more information about these configuration files, see Configuring the Asset Pipeline (p. 269).

Using the Asset Processor Batch Program

The `AssetProcessorBatch.exe` application compiles all assets for the current project and enabled platforms. If the process succeeds without errors, it exits with a 0 code. You can use the Asset Processor Batch program as part of your build system for automation.

The `AssetProcessorBatch.exe` file accepts the following command line parameters for overriding the default behavior:

- `/platforms=` comma separated list
- `/gamefolder=` name of game folder

Example:

```
AssetProcessorBatch.exe /platforms=pc,ios /gamefolder=SamplesProject
```

Asset Processor Interface

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The Asset Processor interface provides areas containing detailed information about the assets that it processes. These areas are shown in the following example.

A – Processing status

Displays asset processing status, the project name, its root location, and the processor port.

B – Tabs

Displays scoped views of Asset Processor functionality. Tabs are described in detail in the following sections.

C – Asset Status

Displays the asset status and details about the source asset in a sortable table.

D – Asset status search with filter

Filters your search by keyword, regular expressions, or status.

E – Event log details

Provides additional information for the selected row in the asset status table.

F – Event log details filter

Filters by warnings and errors.

G – Context details

When enabled, shows specific debugging information for the asset. This setting is disabled by default.
H – Event log line details table

Displays debugging values set in the program for the selected line in the Event log details. These values are emitted during processing.

I – Asset processing timer

After asset processing completes, displays information about the time that it took to perform scans, analyze assets, and produce a build.

Jobs

The Jobs tab displays information about the current state of asset processor jobs. This includes information on the state of jobs, events that occurred during processing, and detailed logging information from the asset processor.

Asset Status

The Asset Status table displays the status of assets that Lumberyard is processing. Select a row to display contextual asset processing information in the Event Log Details table.

The columns in the Asset Status table display the following information about each asset:

- **Source** – Asset's file name and location
- **Completed** – Timestamp of the process completion
- **Platform** – Game platform
- **Job Key** – Specific job process

![Asset Status Table]

Status Column

In the Status column, you can sort by status types.

The column headers display the following information about each asset:

- **Completed** – Completely processed and converted for use in the game
- **Failed** – Failed and needs investigation and debugging
- **In Process** – Currently processing and will display Completed or Failed when complete
- **Pending** – Awaiting processing

Filtering by Keyword and Status

In the filter box, you can filter the table view by entering keywords and regular expressions. The regular expressions are standard `std::regex` in extended format. The standard `std::regex` rules apply.
Example

A regular expression (regex) to search for all files ending with .png:

\*\..png

The asterisk (*) indicates any character 0 or more times.

Example

A regex to search for any files under a \Slices subdirectory:

/\Slices/\.+ 

The dot plus (+) indicates any character 1 or more times.

To view all assets with a particular status, do one of the following:

- Type the status keyword (for example, "failed").
- Click the filter icon and select the status types that you want to see.

Filters that you apply remain active until you remove them. You can either clear the status type box, or click the X next to the filter’s label.

Using a Row’s Context Menu

You can perform actions on each row in the Asset Status table. Right-click on the row to expose a context menu with the following actions:

Show in Asset Browser

Highlights the asset in the Editor's Asset Browser. The Editor must be open and running.

Open in Explorer

Opens the asset in Windows Explorer.

Copy

Copies the asset name.

Open log file

Opens the most recent log file for the asset, if one has been made. File copies and other simple asset processing steps don’t generate logs.

Open folder with log file

Opens the directory with the log file for the asset.
Event Log Details

In the Event Log Details table, you can view asset processing information for an asset that you select in the Asset Status table. This table provides an activity log that shows how the asset was processed and any errors or warnings generated during processing.

The Status column displays icons for the following message types:

- **Information**
  Additional supporting messages are available about the processing of the asset.

- **Warning**
  See the Message column for information about a potential processing issue for this asset.

- **Error**
  The location of the asset and the specific error generated by attempting to process that asset.

The Source column indicates the subsystem that generated the message (such as AssetBuilder) or a generic type, such as Warning or Error.

The Message column displays information related to the processing details of the selected asset.

**Note**
The Message column occasionally prefixes Errors with E: and warnings with W:.

The Event Log Details table also features a context menu to perform copy actions. You can copy the following to the clipboard:

- **Copy Line** – Selected line of the log
- **Copy Line With Details** – Selected line and any related details that appear in the **Event log line details** table
- **Copy All** – All log lines and any hidden details for each item

**Event Log Line Details**

The **Event Log Line Details** table displays when you enable the **Context Details** option. This table displays any additional information about the selected line in the **Event Log Details** table. These details and additional information is generally useful only when debugging issues with a particular asset.

The **Event Log Line Details** table also features a context menu to perform copy actions. You can copy the following to the clipboard:
• **Copy Selected Key** – Text in the **Key** column of the selected row
• **Copy Selected Value** – Text in the **Value** column of the selected row
• **Copy All Values** – All keys and values in the table

Assets

The **Assets** tab displays information about the assets associated with your active project. The data you get includes things like the name of the asset, what files are produced by the resource compiler, and dependencies. In addition to seeing information on source assets, you can also look through generated assets and try to locate missing dependencies.

Source assets

Using the **Source Assets** view of the **Assets** tab shows you the assets picked up during asset processing and lets you investigate their dependencies, products, associated jobs, and force asset rebuilds.
Using Asset Processor

A – Search bar

Search for assets. The search bar is visible in both the Source Assets and Product Assets views. The search will match partial file names and supports regular expressions. Search bar functionality is the same between Source Assets and Product Assets views.

B – Source asset list

The list of all source assets which match the current search filter. When the search is empty, shows all of the source assets used by the project.

C – Asset information

Detailed information about the asset currently selected in the asset list. This includes the name of the asset, the containing folder on the filesystem, and the GUID associated with the asset in the Lumberyard asset system.

D – Products

The compiled asset products that are produced from the source asset. Selecting the popout icon ( ) next to a product name takes you to that asset within the Product Assets view.

E – Outgoing source dependencies

The list of source assets which require an output from this source asset to process. See Why define product dependencies? (p. 337) for more information on product dependencies.

F – Incoming source dependencies

The list of source assets which must have their jobs completed before the processing of this asset. See Why define product dependencies? (p. 337) for more information on product dependencies.
Product assets

A – Product asset list

The list of all product assets which match the current search filter. When the search is empty, shows all of the available products.

B – Asset information

The information for the product asset. Includes the asset GUID, the last time the product was generated, which type of job generated the asset, which platform the asset was produced for, and which source asset is the primary input for the product.

C – Outgoing product dependencies

The list of product assets which depend on this product. In order for your project to function and be distributed properly, all of these assets need to be in the final bundle. See Why define product dependencies? (p. 337) for more information on product dependencies.

D – Outgoing unmet path product dependencies

The list of product assets which are hardcoded paths to be loaded by the Lumberyard runtime that this asset depends upon. Because these products aren’t necessarily generated by the asset processor, they’re placed into a separate category of dependencies. See Hardcoded File Loads (p. 364) for information on resolving path product dependency issues.

E – Incoming product dependencies

The list of product assets which this product depends on. In order for your project to function and be distributed properly, all of these assets need to be in the final bundle. See Why define product dependencies? (p. 337) for more information on product dependencies.
F – Missing dependency scanner

Run the missing dependency scanner from inside the Asset Processor. See Using the Missing Dependency Scanner (p. 366) for more information.

Logs

The Logs tab displays events for the internal operation of the Asset Processor. This area doesn’t display logs for the processing of individual assets. The information in these logs is helpful for troubleshooting the Asset Processor if an issue occurs.

Right-click to access the Logs context menu.

<table>
<thead>
<tr>
<th>Status</th>
<th>Source</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>Scene Builder (BD3BF658-9485-4FE3-88E8-8CE4A13253F3)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>ParticlePreloadLibsBuilderWorker (A00C966-6F96-4510-852-031451235238)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>Slice Builder (B92AD6C-D301-448A-878-8227196A1B5A)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>Image Worker Builder (A00C966-6F96-4510-852-031451235238)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>Script Events Builder (CD64F8A-01F-4D5F-892A-50D83E22D943)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>Atlas Worker Builder (79036168-E01-497-9EC-8D7D509B054)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>Script Canvas Builder (6E86272B-7C0-48A3-A5D-8E89206DF031)</td>
</tr>
<tr>
<td>6/9/2... AssetProcessor</td>
<td>Building...</td>
<td>SchemaBuilderWorker (8BE5B2E93-03)</td>
</tr>
</tbody>
</table>

Shaders

The Shaders tab displays a table with information related to shader compiler proxies. Shader failures appear in this table. For more information, see Shader Compiler Proxy (p. 292).
Connections

The Connections tab displays devices and programs that the Asset Processor is connected to and the platform they are running on. You can add approved connections in the White Listed Connections box and disapproved connections in Rejected Connections.

In the Active Connections table's Enabled column, automatic connections are labeled as Auto. This means that it's a connection that the Asset Processor created. One example of such a connection is Asset Builder connections. User-created connections show a check box. If you select the check box, the Asset Processor continually attempts to reconnect to those connections. You can use these custom connections for specialized cases, such as when connecting to mobile devices outside of a company's internal network.

You can edit or remove a user-created connection, or add a connection.
Tools

On the Tools tab, you control how your assets are scanned. Use Faster Scanning Mode when you don't need to perform a full asset scan. For more information, see Enabling Asset Processor's Faster Scanning Mode (p. 261).

Enabling Asset Processor's Faster Scanning Mode

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Asset Processor's Faster Scanning Mode speeds up Lumberyard's startup scan by disabling checking for cache changes that occurred while Asset Processor was not running. This can save you time when processing many assets in your project.

By default, Faster Scanning Mode is enabled. You can enable or disable this mode any time without restarting the Asset Processor, including during the scan. Asset Processor saves your preference between sessions.

To disable Faster Scanning Mode

1. Open Asset Processor (p. 248).
2. Select Tools and clear Faster Scanning Mode.
To perform a full scan

- Click Start Scan. This starts a full scan immediately, which checks for files missing from the cache and rebuilds the appropriate source files.

Asset Processor's batch version also supports a command line parameter for Faster Scanning Mode.

```
AssetProcessorBatch.exe --zeroAnalysisMode
```

When you enable the feature you can also use console mode stdout as well as open GUI log windows that indicate the effectiveness of Faster Scanning Mode. The output message displays the number of files that required full analysis and the total number of files.

**Example**

```
--Debug--5303 files reported from scanner. 2903 unchanged files skipped, 2400 files processed
```

For more information, see Configuring the Asset Pipeline (p. 269).

**Performing a Full Scan**

When you choose a full scan, Asset Processor performs the following actions. Low cost actions contribute to less than 1% of the total time.

- (Low) Determines which builders are responsible for building a file. Asset Processor examines builder patterns, such as *.tif or *.fbx when they register.
- (Low) Lookup the file in the database to determine what happened to it the previous time.
- (Low) Compares the new job fingerprints to the previously emitted jobs to check for differences.
- (Moderate) Generates a job fingerprint that includes modtimes of source files and dependencies and the versions of the involved builders.
- (High) Checks the cache directory and ensures that every product that was emitted last time for this source file is still present.
- (Very high) Sends the file to the registered builders so that they can spawn jobs for the job queue.

If all of the following are true for a given source file found during the scan, then that file can be excluded from reanalysis.

- The source file hasn't changed on disk (it has the same modtime as previously).
- The files on which the source file depends haven't changed on disk (it has the same modtime as previously).
- The builders assigned to that source file successfully processed it last time. This means there were no failures or errors.
- The builders assigned to that source file didn't get a new version or a new analysis fingerprint.
- There are no new builders that may operate on that source file.
- There are no builders that were previously enabled which could have operated on that source file but have since been removed.
- There are no builders that have changed the set of source files.

Asset Processor performs a check at the beginning of the scan and does the following actions. These actions are low cost.
Using Asset Processor

• (Very low) Collects modtime information for every file scanned as part of Asset Processor startup.
• (Very low) Checks whether the source file in question had any builders version change or analysis fingerprint by comparing the data from the previous time.
• (Very low) Compares the modtime with the previously recorded time in the database.
• (Low) Queries the database files table to get a list of every file and its modtime from previous runs.
• (Low) Queries the database sources table to get a list of every source asset and its builder-fingerprint signature.

If any of these checks fail, the source goes through the normal, unchanged analysis pipeline. This means that Faster Scanning Mode makes no changes to the actual analysis.

Importing Assets into Lumberyard

You can import assets into your Lumberyard game project directory with one of the following.

Importing Assets Manually into Lumberyard

You can import assets even if Lumberyard isn't running. You can do this at any time.

To import assets manually
1. Navigate to your game project directory, such as `lumberyard_version/dev/My_Game_Project/Objects`.
2. Copy or move the assets to your game project directory. For example, you can add your `.cgf` files to the specified directory.
   • If Asset Processor is running, it detects the new files and converts them for use in Lumberyard Editor.
   • If Asset Processor isn't running, Asset Processor automatically detects the new files the next time you start Lumberyard Editor.

Importing Assets from Lumberyard Editor

To import assets from Lumberyard Editor
1. In Lumberyard Editor, choose File and then Import.
2. Navigate to your assets and select the files to add to your game project directory. Then click OK.
3. In the Import Asset(s) dialog box, the Destination Folder path automatically shows the root of your game project directory. You can click Browse to specify a subdirectory in your game project directory.
   
   Note
   You can import assets to a directory only in the current game project.
4. Select whether you want to Copy Files from their original location or Move Files and then click Import. If you copy the files, Asset Processor doesn't monitor changes to the original files.
5. In the **Processing asset** dialog box, click **View status** to check the status of your assets.

**Tutorials: Importing Assets into Lumberyard**

To learn more about importing your assets, see the following video tutorials:

- Tutorials: Importing FBX Files as Actors and Motions (p. 1497)
- FBX Settings mesh export (p. 436)

**Using Resource Compiler**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Resource Compiler converts certain assets into a format that Lumberyard can use. Resource Compiler stores the converted, game-ready assets in a cache that mirrors your project directory. This cache is located in the \lumberyard_version\dev\Cache directory. Resource Compiler can also create PAK files.
Asset Processor calls Resource Compiler to automatically convert new or modified assets in your Lumberyard directory. The Resource Compiler application, `rc.exe`, is located in the `\lumberyard_version\dev\Bin64vc141\rc` directory.

Resource Compiler converts the following file types.

**ABC**
- Alembic files

**CGF**
- Legacy geometry format

**I_CAF**
- Legacy animation format

**TIF, BMP, JPG, PNG**
- Image files (p. 278)

**FBX**
- Geometry (static mesh) only. Doesn't convert new EMotion FX characters or animations.

**Note**
All other types, such as characters and animation, are converted by the BuilderSDK system. For more information, see Creating a Custom Asset Builder (p. 318).

You can customize how resources are converted by modifying the `AssetProcessorPlatformConfig.ini`. For more information, see Configuring the Asset Pipeline (p. 269).

You can access a full list of arguments available for use in `AssetProcessorPlatformConfig.ini`. To access a full list of arguments available for the `rc.exe` command line tool:

1. Open a command line.
2. Navigate to the `\lumberyard_version\dev\Bin64vc141\rc` directory.
3. Enter the following command:
   
   ```
   rc /help
   ```

   To output the help text to a file, enter the following command:

   ```
   rc /help >file.txt
   ```

**Debugging Asset Processor**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the **AWS Game Tech blog** to learn more.

Use the following options to help debug Asset Processor issues.
Topics

- Viewing Asset Processor Log Files (p. 266)
- Restarting Asset Processor (p. 267)
- Using Asset Builder (p. 267)
- Using the Microsoft Child Process Debugging Power Tool (p. 268)
- Debug Asset Builders from Asset Processor (p. 269)
- Clearing the Cache (p. 269)

Viewing Asset Processor Log Files

You can view logs for internal operations of Asset Processor. If Asset Processor isn’t processing or working as expected, use the information in the logs to debug the issue. This doesn’t include logs for the processing of individual assets.

To view the Asset Processor log files

1. In Asset Processor, choose Logs.
2. In the Logs section, you can view the following:
   - Status – The date and time stamp of the log
   - Source – Where the log came from (for example, Asset Processor)
   - Message – The description of the log

Example

3. To create another log report, click Add.
4. In the Create New Logging Tab, you can specify the following settings.
   - Filter name – Enter the name of your filter (for example, All logs).
   - Text filter (optional) – Enter text to filter the log results.
   - Show messages – Displays messages about each log.
   - Show warnings – Displays logs that have warnings.
   - Show errors – Displays logs that have errors.
   - Show debug – Displays logs that have debug issues.
5. Click OK. Your log report appears as another tab in Asset Processor.
6. You can click Copy all and paste the raw logs into a text file.
7. You can also click Open log files to open \lumberyard_version\dev\Bin64vc141\logs\JobLogs directory.
8. In a text editor, open the example.log file.

### Restarting Asset Processor

You can restart Lumberyard Editor and Asset Processor. Verify that only one instance of Asset Processor runs at the same time.

**To restart Asset Processor**

1. Close Lumberyard Editor.
2. In the Windows taskbar, right-click Asset Processor, and choose Quit or press Ctrl+Q.
3. Restart the project or branch that you are working on. Asset Processor automatically starts.

### Using Asset Builder

You can also debug Asset Processor using Asset Builder. This is a standalone AzToolsFramework application that lets you run BuilderSDK modules in isolation. You can run AssetBuilder in debug mode to develop new features for a builder. In debug mode, Asset Builder creates a test job or processes jobs for specified files.

**Note**

You must start Asset Processor before you can enter a -debug command.

**To debug Asset Processor using Asset Builder**

1. Navigate to the \lumberyard_version\dev\Bin64vc141 directory.
2. In a command line prompt, enter the following command to get a list of possible options.

   ![AssetBuilder.exe -help](AssetBuilder.exe -help)

3. You can use the following debug options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-debug</td>
<td>Creates and processes jobs for the specified file.</td>
</tr>
</tbody>
</table>
Using Asset Processor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-debug_create</td>
<td>Creates, but doesn't process jobs for the specified file.</td>
</tr>
<tr>
<td>-debug_process</td>
<td>Processes, but doesn't create jobs for the specified file.</td>
</tr>
<tr>
<td>-output</td>
<td>The directory to write the result to. If you don't specify the directory,</td>
</tr>
<tr>
<td></td>
<td>the default directory is Debug*filename in the builders binary directory.</td>
</tr>
<tr>
<td>-input</td>
<td>If you enable debug mode using the -task option, you must use -input with</td>
</tr>
<tr>
<td></td>
<td>the specified file.</td>
</tr>
<tr>
<td>-tags</td>
<td>The list of additional tags to append to the platform tag list. You can use</td>
</tr>
<tr>
<td></td>
<td>tags to replicate the same list of platform tags that the source asset uses</td>
</tr>
<tr>
<td></td>
<td>in the normal Asset Processor workflow.</td>
</tr>
</tbody>
</table>

**Example**

To debug a specified file, run the following command.

```
AssetBuilder.exe -debug Objects\Tutorials\Fbx\shapes.fbx
```

**Example**

To create a job without processing a specified file, run the following command.

```
AssetBuilder.exe -debug_create "Objects\Tutorials\Fbx\shapes.fbx" -module "C:\lumberyard_version\dev\Bin64vc141.Debug\Builders\ExampleBuilder.dll" -output "C:\lumberyard_version\dev\Logs\Shapes\"
```

**Example**

To process without creating a job for a specified file, run the following command.

```
AssetBuilder.exe -debug_process "Objects\Tutorials\Fbx\shapes.fbx"
```

**Using the Microsoft Child Process Debugging Power Tool**

Use this tool to automatically attach the debugger to spawned child processes.

**To use the Microsoft Child Process Debugging Power Tool**

1. Go to the download page, and click Download.
2. Install the tool for Visual Studio.
**Debug Asset Builders from Asset Processor**

Use the following procedure to debug in the following scenarios:

- Debug intermittent failures that are difficult to reproduce in a single run of Asset Builder using the –debug option
- Debug failures that only occur in multiple process job requests

**To debug Asset Builders from Asset Processor**

1. Navigate to the `lumberyard_version\dev` directory.
2. In a text editor, open the `AssetProcessorPlatformConfig.ini` file and set `maxjobs=1`. This limits Asset Processor to run one job at a time.
3. Run Asset Processor so that it spawns the Asset Builder process.
4. To debug, attach the `AssetBuilder.exe` in Visual Studio. There is only one Asset Builder.
   
   The next time that you modify your source file, `AssetBuilder.exe` builds that asset.

**Tip**

You can spawn multiple instances of `AssetBuilder.exe` and attach them to Visual Studio.

**Clearing the Cache**

If you're a game artist and you're having issues running Asset Processor, this might be the result of a corrupt cache. In this case, you can delete your `Cache` directory. When you delete your cache, you can restart Asset Processor to rebuild all of your assets.

**Note**

- If you're an engineer making new BuilderSDK-based builders, we recommend that you don't delete your cache.

**To delete the Asset Processor cache**

1. Quit Asset Processor.
2. Navigate to the `lumberyard_version/dev/Bin` directory.
3. Delete the `Cache` directory.
4. Restart Asset Processor to rebuild all assets.

**Configuring the Asset Pipeline**

**Important**

The Asset Builder SDK is now preferred over the legacy `rc.exe` program for adding asset types to the pipeline. Instead of using the `rc.exe` program, make a builder module that you derive from the BuilderSDK. These modules are self configuring. For instructions and examples on how to write builders that process your own asset types, see the [Creating a Custom Asset Builder (p. 318)](#). We recommend that you do not rely on the old `rc.exe` pipeline, although it's still available if you have legacy code.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.
You can configure the Lumberyard asset pipeline by editing the `\dev\AssetProcessorPlatformConfig.ini` file that rc.exe program uses. You can add your own asset types to it by modifying the sections of the file described in this document. When you check in your changes to the config file, the version of the assets on your collaborators’ computers is updated automatically. This removes the need for you to manually refresh the cache on each coworker’s computer.

The `AssetProcessorPlatformConfig.ini` consists of six sections. The `.ini` file uses standard Qt/Windows `.ini` file formatting rules. Comments are preceded by a semicolon, and named sections are designated by square brackets.

**Important**

Backslashes in `.ini` files have a special meaning. To use a regular backslash character, you must prefix it with another backslash. To avoid problems with file paths, the asset processor and asset pipeline use forward slashes for path names. However, if you need to use backslashes in regular expressions, you must also escape them so that they can be recognized by the regex system. For example, you must specify the regular expression `.*/\Levels\).* like this:

```
.*\/\Levels\\/\.*
```

In addition to the `AssetProcessorPlatformConfig.ini` file (located in the `/lumberyard_version/dev` directory), you can also add the following:

- `AssetProcessorGamePlatformConfig.ini` – Add this file to your game project folder to override any configurations that are specific to a project. The final configuration is the result of merging both files. The `AssetProcessorGamePlatformConfig.ini` file is read last and therefore takes priority.
- `AssetProcessorGemConfig.ini` – Add this file to the gem folder to allow your enabled gem to affect the asset processor configuration. The impact is similar to editing the root file but without making permanent changes to the root file. For example, you can add an `AssetProcessorGemConfig.ini` file for the Cloud gem to the `/lumberyard_version/dev/Gems/Clouds` directory.

The final configuration is the result of merging the following files in order:

- `dev/AssetProcessorPlatformConfig.ini`
- `dev/Gems/GEM_NAME1/AssetProcessorGemConfig.ini`
- `dev/Gems/GEM_NAME2/AssetProcessorGemConfig.ini`
- `dev/Gems/GEM_NAME3/AssetProcessorGemConfig.ini`
- `your_project_folder/AssetProcessorGamePlatformConfig.ini`

**Platforms Section**

The `Platforms` section contains two subsections:

- **Platform Definition** – Defines which platforms exist when you use the `[Platform platformName]` section header.
- **Platforms** – Defines which of the platforms are enabled by default in your project.

**Platform Definition Subsection**

Use the following subsections to define an operating system and its attributes. The name of the operating system appears in the section header, for example, `pc` or `ios`.

```
[Platform pc]
tagstoools,renderer
```
You can define the tags attribute and assign tags to an operating system. This allows you to control behavior based on the tags. For example, you can choose to compile textures on all operating systems with the mobile tag rather than naming operating systems individually. Should you add mobile operating systems to your configuration, you can use the mobile tag to include them when modifying behavior. This removes the necessity of recompiling and modifying all rules and builders to include the new operating systems.

**Platforms Subsection**

Use the Platforms subsection to enable and disable operating systems for the entire project. Note that "disabling" means that the game project does not use the specified operating system. When you disable an operating system, the related assets are removed, and the associated space on your hard drive is freed up.

In the following example, PC is enabled and other operating systems are commented out.

```
[Platforms]
pc=enabled ;es3=enabled ;ios=enabled ;osx_gl=enabled
```

Because the default value for an operating system is disabled, the operating systems in the example that are commented out are not enabled.

If you want to enable an operating system that is already listed in the [Platforms] subsection, remove the semicolon to uncomment the corresponding line.

If an entry for a game operating system that you want is not in the list, you can add it. However, you must also handle OS-related differences like image formats. To do so, you must change the code in the asset processor (and possibly the image compiler and other builders).

If you are using the rc.exe pipeline, specified operating systems are passed as parameters to the rc.exe program.

The operating system that Lumberyard runs on is enabled by default, so you can leave that line commented out. For example, if Lumberyard runs on Mac, you can leave the osx_gl=enabled line commented out.

If you run AssetProcessor.exe or AssetProcessorBatch.exe on a build server, you can use the following command line parameter to specify which operating systems to enable: /platforms=comma-separated-list

Using this command line parameter enables only the specified operating systems, regardless of the host platform that runs the tools.

You can specify tags for the platforms command line parameter, for example:

```
AssetProcessorBatch.exe /platforms=tools,es3
```

**Jobs Section**

Use the Jobs section to control how many parallel jobs to run, as in the following example.

```
; ---- The number of worker jobs, 0 means use the number of logical cores
[Jobs]
```
minJobs=1
maxJobs=0

Setting `maxJobs` to zero specifies using as many cores as are available. A number other than zero limits the cores used to no more than the number that you specify.

**MetaDataTypes Section**

Use the `MetaDataTypes` section to tell the asset system that certain file types are associated with other files in the same folder. These specifications control the compilation of side-by-side assets, as in the following example.

```plaintext
[MetaDataTypes]
eventsettings=
animsettings=i_caf
Animations/SkeletonList.xml=i_caf
cbc=abc
fbx.assetinfo=fbx
```

Entries on the left and right sides of the equals sign specify file extensions of asset files in the same folder. If a file with the extension on the left changes, then the file with the extension on the right must also be rebuilt if it has the same filename. For example, the line `animsettings=i_caf` means that if a file called `example.animsettings` changes, `example.i_caf` will be recompiled.

The line `exportsettings=` means that when any file with the extension `.exportsettings` changes, any asset file that has the same filename as the file with the `.exportsettings` extension is invalidated. For example, a change in the `MyImage.TIF.exportsettings` file invalidates the `MyImage.TIF` file.

In the example `Animations/SkeletonList.xml=i_caf`, the left side specifies not an extension, but a specific file. Whenever the `Animations/SkeletonList.xml` file changes (note that the forward slash indicates a directory path), all files with the extension `.i_caf` are invalidated.

**Note**

If you use the Asset Builder SDK, you can declare your dependencies on other files explicitly. This makes the `[MetaDataTypes]` section less important.

**ScanFolder Section**

Use the `ScanFolder` section to direct the Asset Processor to monitor the assets in specific folders. The following example directs the Asset Processor to monitor the `Editor` folder.

```plaintext
[ScanFolder Editor]
watch=@ROOT@/Editor
output=editor
recursive=1
order=30000
```

You can add as many scan folders as you want, but each folder must have a unique name. Because the scan folders are stored in a hash table using the name specified in square brackets, make sure that the name following `ScanFolder` is unique.

You can use the aliases `@root@` and `@gamename@` as placeholders to enable portability to the computers of other users who are working on the same project.

To make the scan folder OS-specific, use the keywords `include` and `exclude`. Both keywords can contain platform tags and/or platform identifiers. If you do not specify a keyword, all enabled operating systems are included by default.

The `ScanFolder` section has the following parameters.
### Configuring the Asset Pipeline

<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>watch=&lt;foldername&gt;</td>
<td>Watch this specific folder for assets.</td>
</tr>
<tr>
<td>output=&lt;foldername&gt;</td>
<td>Put the contents of this watch folder into the subfolder of the @assets@ folder called &lt;foldername&gt;.</td>
</tr>
<tr>
<td>recursive=1</td>
<td>Recurse into subfolders.</td>
</tr>
<tr>
<td>order=30000</td>
<td>Declares a priority order. The lower the number, the more &quot;important&quot; a folder is. The game folder for your project is always considered 0, the most important.</td>
</tr>
</tbody>
</table>

**Note**

The *order* parameter affects only assets with the same name. For example, suppose you have an asset called MyTexture.TIF in two separate scan folders. If both asset files map to the same output file, then the asset file with the lower order number overrides the one with the higher.

| include=<comma-separated platform tags or identifiers> | Contains the list of platforms or platform tags to include for the scan folder. Only enabled platforms are included. If you include a disabled platform, it will not be considered for the scan folder. |
| exclude=<comma-separated platform tags or identifiers> | Contains the list of platforms or platform tags to exclude for the scan folder. |

### Notes

- In most cases, you do not need to specify an output folder. The output folder remaps source folders into subfolders of the cache. Usually folders that contain assets go into the cache directly, without requiring a subfolder.
- It is not considered an error if a scan folder is missing. This behavior is by design because it lets you have optional folders for assets. For example, this might be useful for test cases.
- Removing folders from the ScanFolder sections removes any corresponding assets from the cache. If the assets specified were overriding other assets, the overridden assets are reinstated and become primary assets again.

### Exclude Section

Use the Exclude section to add file path patterns to ignore. As in the rest of the ini file, backslashes must be prefixed with an extra backslash to escape them from .ini file processing.

The following example excludes alembic compression templates and temporary animation compression files.

```ini
[Exclude AlembicCompressionTemplates]
pattern=.*Presets\/GeomCache\/.*

[Exclude TmpAnimationCompression]
pattern=.*Editor\/Tmp\/AnimationCompression\/.*
```

### Notes
- The regular expressions are standard `STD::regex` in extended format. Standard `STD::regex` rules apply.
- The input paths are always absolute paths. If you don't want to filter by absolute path, start your regular expressions with `.*`, as in the example.
- If you want to add new exclude rules, give them a unique name. The actual name does not matter as long as each is unique.

**RC Section**

Use the RC section to specify files to be processed by the `rc.exe` program or to be copied as-is into the asset cache without processing. The RC section is only for use by legacy RC modules and for specifying simple file copies to cache.

The RC section consists of a series of recognizer descriptors. Each descriptor specifies a set of files (by glob or by pattern) and what to do with the specified files. Changing the fields of the recognizer invalidates assets according to the change made.

**Important**

Because they do not use the legacy `rc.exe` program, builders implemented as builder modules do not use the RC section. Instead, they derive their configuration programmatically or read it from a custom config file. If you create your own BuilderSDK builder, do not add anything to the RC section.

The following code block shows the syntax of the RC section.

```
[RC (recognizer name)]
; ---- Choose either pattern or glob. You cannot choose both.
pattern=(pattern to use to recognize these files)
glob=(glob to use to recognize these files)
(params)=command line params/copy/skip
(platformname)=(params)
lockSource=(true/false)
priority=(0...n); Higher numbers are more important.
critical=(true, false)
version=(0...n)
```

The following table describes each parameter and its options.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>A regular expression that specifies the files to process. When you use regular expressions, remember to escape any backslashes.</td>
</tr>
<tr>
<td>glob</td>
<td>A wildcard expression like <code>*.tif</code> that specifies binary glob files to process.</td>
</tr>
<tr>
<td>params</td>
<td>The <code>params</code> parameter can take one of the following three options.</td>
</tr>
<tr>
<td></td>
<td>1. The default parameters to pass to the <code>rc.exe</code> program to process the kind of asset specified.</td>
</tr>
<tr>
<td></td>
<td>2. <code>copy</code> – Copies the file as-is into the cache. It does not invoke <code>rc.exe</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Copy jobs are the most common type of job. For example, in the <code>dev \assetprocessorplatformconfig.ini</code> file that is included with Lumberyard, most <code>[RC]</code> sections specify <code>params=copy</code>.</td>
</tr>
<tr>
<td></td>
<td>3. <code>skip</code> – Skips the specified file type entirely. The <code>skip</code> option is typically more useful in the <code>platformname</code> parameter.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>If you omit the params, platform params, or tag params sections, the <code>rc.exe</code> program processes the file using default options. To process the file on specific operating systems, set the <code>params</code> parameter to <code>skip</code> and then specify the desired platforms.</td>
<td></td>
</tr>
<tr>
<td>If you pass parameters to the <code>rc.exe</code> program, you can pass <code>p</code> (as in <code>/p=pc</code>) to force the program to process the asset as if it were the PC. You can pass the appropriate parameter for the desired platform.</td>
<td></td>
</tr>
<tr>
<td><code>(platformname)=(params)</code></td>
<td>Specifies OS-specific parameters. You can use the <code>params</code> parameter to specify default parameters, and then override them for specific operating systems when required by using <code>platformname=params</code>. For example, the statement <code>pc=/TEST</code> overrides the default parameters for PC and passes the parameter <code>/TEST</code> to the <code>rc.exe</code> program. Can be <code>true</code> or <code>false</code>. When <code>true</code>, causes the job to wait until it can gain an exclusive read/write lock on the source file.</td>
</tr>
<tr>
<td><code>(tagname)=(params)</code></td>
<td>Specifies tag-specific parameters, which are generally better than platform-specific parameters. For example, the following statement causes all platforms to use the default parameters when processing <code>.tiff</code> files. However, a platform with the <code>server</code> tag is skipped.</td>
</tr>
<tr>
<td><code>server=/p=pc</code></td>
<td>The following statement passes <code>/p=pc</code> to invocations of the <code>rc.exe</code> program: <code>server=/p=pc</code>. This allows the program to process assets (such as textures) as if it were the PC, even on the server. The default behavior is to pass <code>/p=server</code>, which the <code>rc.exe</code> program may not understand. The <code>lockSource</code> parameter is useful for dealing with applications that hold onto a file and then slowly stream data into it. For example, if a program creates very large files over a long period of time, you can set <code>lockSource=true</code> to avoid processing an asset until the other application releases it. Use of this parameter is relatively rare and is generally expensive, so you should avoid using it unless absolutely required.</td>
</tr>
<tr>
<td><code>priority</code></td>
<td>Specifies job priority. A larger number gives a job greater priority in the queue. Normally, you should assign a larger number to assets that are likely to be needed from the start or that affect gameplay. This ensures that they get compiled sooner. Note params copy jobs have an default priority of 1.</td>
</tr>
</tbody>
</table>
### Configuring the Asset Pipeline

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| critical  | Can be true or false. Critical jobs cause the editor splash screen to continue displaying and pause the startup of the runtime until every critical job has been completed. Marking jobs as critical ensures they are complete before the editor is allowed to start. You can specify entire types of assets as critical. This can be useful for files that are used during startup, cause bad behavior if they are not ready during bootstrap, or cannot be reloaded live. **Important** Because critical jobs can delay the startup of the editor for the first time, not having critical jobs is always the preferred choice. Alternative approaches include:  
  • Making the editor or runtime capable of reloading the asset live after it is compiled.  
  • Making a call to compile the asset on demand using the asset system bus. You can use the public function CompileAssetSync to do this. See the Lumberyard source code for examples.  

  **Note** params copy jobs are critical by default.  
| version   | An arbitrary versioning number. The default is 0. Changing the version number invalidates the assets specified and causes them to be rebuilt. The version parameter provides a convenient way to cause a rebuild of all assets of a particular kind. For example, you might make changes to the compiler that builds a particular kind of asset. Then, when you check in your changes to the .ini file, local assets of workers receiving the update are rebuilt for them automatically.  

The following example specifies how .tiff files are to be processed.

```ini
[RC tif]
pattern=.*\.tiff?
params=/imagecompressor=CTSquish /streaming=1
es3=/imagecompressor=CTSquish /streaming=0
ios=/imagecompressor=CTSquish /streaming=0
; Streaming = 1 splits files.
lockSource=true
```

The example has the following characteristics.

- It declares a recognizer called tif (because [RC tif] is the name).
- The pattern specifies all files which match the regex expression .*/\.tiff?. Note that the example escapes the backslash.
- The params parameter specifies the default parameters with which to invoke rc.exe. In the example, .tiff files will be compiled into the /imagecompressor=CTSquish /streaming=1 format.
- For ES3 and iOS, streaming is turned off, overriding the default that was specified in the params parameter.
- lockSource is set to true to avoid conflict with external tools that create a zero byte file, pause for many seconds, and then fill it with data.
The following example specifies how .tiff files in the GoldenImages subfolder are to be processed.

```ini
; Feature tests use the raw .tif files for the golden image comparison.
[RC goldenimages]
pattern=.*GoldenImages\/*.\*.tif
params=copy
```

The example has the following characteristics.

- It declares a recognizer called goldenimages which applies to any .tiff file in the GoldenImages subfolder.
- The params parameter specifies copy, so any .tiff file in the GoldenImages subfolder is copied to the cache without processing.

**Notes**

The two example RC sections are both in the same file. This has the following important consequences:

- The multiple rules that match the files all apply simultaneously. They are not exclusive. If you have two rules that apply to the same file, both rules are run. For example, the rules in the two examples would both apply to a file called \dev\SamplesProject\textures\GoldenImages\myfile.tif. The rules would produce both a .dds compressed version of myfile.tif and an uncompressed myfile.tif file that is copied into the cache.
- If you want to specify an exclusive subfolder rule, you must use inverse regex selectors to create exclusion patterns.

The following example shows a set of rules that apply exclusively to .png files. The two rules are written so that any .png file matches only one of the rules.

```ini
;Example: Use the specified parameters to process all .png files except those in the libs/ui folder.
[RC png-normal]
pattern=(?!.*libs\/ui\/).*\.png
params=/imagecompressor=CTSquish /streaming=0
lockSource=true

;Example: Process all .png files in the libs/ui folder using linear color space.
[RC png-ui]
pattern=(.*libs\/ui\/).*\.png
params=/imagecompressor=CTSquish /streaming=0 /colorspace=linear,linear
lockSource=true
```

For more examples, see the default \dev\AssetProcessorPlatformConfig.ini file.

**Common Problems**

When troubleshooting, be aware of the following pitfalls.

- Not escaping your regular expressions with two backslashes. Remember that one of the slashes is removed when the .ini file is processed.
- Duplicating a rule without changing its name. The rule [RC png] has a name of png. These names are inserted into an unordered hash. If you specify another section with the name of png, the second section overwrites the other in random order. This behavior is by design. For example, you can use it to allow your game version of the .ini file to override particular sections or specify skip to skip them. Otherwise, if you want to add new rules, give them a unique name. The actual name does not matter.
as long as each is unique. This is especially true for named sections like the Exclude and ScanFolder sections.

- Not understanding that all recognizers that match apply, not just the first one.
- Forgetting to prefix your regular expressions with .*. By default, the input files that you specify are considered absolute paths. This behavior is by design because it lets you exclude or include files based on absolute paths, if that is your intent. Use the .* prefix if you want to use relative paths.

### Configuring Image Processing

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Images or textures are automatically processed by Asset Processor, which makes them ready for Lumberyard game creation. When you place an image file anywhere within your Lumberyard directory, Asset Processor detects and converts the file to a game-ready asset. Lumberyard has several configuration files that specify settings for the conversion process, such as the colorspace to use, texture size, whether to generate mip maps (p. 286), and so on.

You can edit the configuration files (p. 280) to create your own image processing presets. To use these presets, you append existing or customized suffixes to your image file names. Based on these file name suffixes, Asset Processor automatically uses the appropriate preset to convert the image.

After Asset Processor successfully converts the image, the resulting .dds asset is placed in the appropriate lumberyard_version\dev\Cache directory. The image source file remains in its original form and location.

Asset Processor calls the Resource Compiler, rc.exe (located in lumberyard_version\dev\Bin64vc141\rc), to convert the image files. Resource Compiler also has an image tool (p. 283), which you can use to manually select image presets, mip maps, and so on. The Resource Compiler image tool can also output an imagefilename.exportsettings, which you must check in if your project uses source control such as Perforce. Checking in the .exportsettings file ensures that the texture renders similarly for each person working on the project.

Lumberyard supports the following image file types:

- TIFF – No grayscale
- TGA – Indexed or true color only; no grayscale
- PNG – RGB or RGBA only; no grayscale
- BMP – Any format
- GIF – Supported, but not an ideal texture format
- JPG – Any format

The following diagram represents the general workflow for converting images.
Topics
• Creating Image Processing Presets (p. 280)
Creating Image Processing Presets

Lumberyard contains an existing set of image processing presets. You apply these presets on your source image file names before you add them in your Lumberyard directory for processing. Based on the appended suffix for the file name, Asset Processor converts them into .dds files with specific settings and adds them to the `lumberyard_version\dev\Cache` directory.

You can define presets in two ways:

(Recommended method) **ImageCompiler.cpp and rc.ini**

The `ImageCompiler.cpp` file contains the code that finds a file name suffix match, and points to another file for the specific settings. The `rc.ini` file defines those specific settings for the matched suffix.

**AssetProcessorPlatformConfig.ini**

The `AssetProcessorPlatformConfig.ini` file matches file name suffix patterns and then calls functions to process the file based on its matches.

Topics

- Creating Presets with ImageCompiler.cpp and rc.ini (p. 280)
- Creating Presets with AssetProcessorPlatformConfig.ini (p. 282)

Creating Presets with ImageCompiler.cpp and rc.ini

The following snippet is from `ImageCompiler.cpp`, located in the `lumberyard_version\dev\Code\Tools\RC\ResourceCompilerImage` directory. Part of this code finds file names appended with `_bump`, and then calls the `presetName` function `defaultBump`, which derives its settings from `NormalsFromDisplacement` (defined in `rc.ini`).

```cpp
... static string AutoselectPreset(const ConvertContext& CC, const uint32 width, const uint32 height, const bool hasAlpha) {
    const char* const defaultColorchart = "ColorChart";
    const char* const defaultBump = "NormalsFromDisplacement";
    const char* const defaultNormalmap = "Normals";
    ...
    const string fileName = CC.config->GetAsString("overwritefilename", CC.sourceFileNameOnly, CC.sourceFileNameOnly);
    string presetName;
    if (SuffixUtil::HasSuffix(fileName.c_str(), '_', "cch"))
    {
        presetName = defaultColorchart;
    }
    else if (SuffixUtil::HasSuffix(fileName.c_str(), '_', "bump"))
    {
        presetName = defaultBump;
    }
...```
Configuring Image Processing

```cpp
else if (SuffixUtil::HasSuffix(fileName.c_str(), '_','ddn'))
{
    presetName = defaultNormalmap;
}
...
```

The `rc.ini` file (located in the `lumberyard_version\dev\Bin64vc141\rc` directory) contains the following setup information:

- Asset presets for operating systems.
- Image preset aliases (profile mapping from a legacy preset).
- Image format settings, such as compression method, pixel format, and so on.
- Mip map settings.
- Naming convention presets: Image suffixes such as `_diff`, `spec/refl`, `_disp`, `_ddna`, `_ddn`, `_bump`, `cm/cubemap`, `detail`, and so on.

**Example**

The following snippet from the `rc.ini` file defines `NormalsFromDisplacement`. It includes pixel formats for each operating system, maximum texture size, mip map settings (p. 286), colorspace, and so on.

```ini
... ; converts greyscale texture to normal map, normal map textures (signed BC5)
[NormalsFromDisplacement]
pixelformat=BC5s
;pixelformat:es3=ASTC_5x5
;pixelformat:es3=EAC_RG11
;pixelformat:ios=PVRTC4
pixelformat:ios=EAC_RG11
maxtexturesize:es3=1024
maxtexturesize:ios=1024
bumpertype=1
powof2=1
mipnormalize=1
mipmaps=1
mipgentype=average
colorspace=linear,linear
filemasks=_bump*
...
```

You can edit `ImageCompiler.cpp` and `rc.ini` to add your own image presets.

**To add your own image processing presets**

1. In a text editor, open `ImageCompiler.cpp`, located in the `lumberyard_version\dev\Code\Tools\RC\ResourceCompilerImage` directory.
2. Specify the appropriate lines of code to define your new preset.
3. Save and close `ImageCompiler.cpp`.
4. In a text editor, open `rc.ini`, located in the `lumberyard_version\dev\Bin64vc141\rc` directory.
5. Add your new definition.
6. Specify the settings for the new definition.
7. Save and close `rc.ini`. 

Version 1.28
281
Example

If you want to add a line of code that searches for file names that end in `_awesome`, you might add the following lines to your `ImageCompiler.cpp`.

```c++
const char* const defaultAwesomeSuffix = "AwesomeImage";
```

Example

```c++
else if (SuffixUtil::HasSuffix(fileName.c_str(), '\', "awesome"))
{
    presetName = defaultAwesomeSuffix;
}
```

You would also add to a definition of `AwesomeImage` to the `rc.ini` file.

Example

```
[AwesomeImage]
pixelformat=BC1
pixelformat:es3=ETC2
pixelformat:ios=FVRTC2
;pixelformat:es3=ASTC_4x4
;pixelformat:ios=ASTC_6x6
rgbweights=ciexyz
pwof2=1
mipmaps=1
colorspace=sRGB,auto
discardalpha=1
filemasks=_*_awesome*
```

Creating Presets with `AssetProcessorPlatformConfig.ini`

You can also add presets in the `AssetProcessorPlatformConfig.ini` file, located in the `lumberyard_version\dev` directory.

Example

The following snippet from `AssetProcessorPlatformConfig.ini` shows two rules, both of which match a file that ends in `_awesome.tif`. The image is processed for each rule that it matches. In this example, multiple results (assets) would be produced for each image.

```ini
...[RC uitextures]
pattern=.*\.(bmp|gif|jpg|jpeg|tga|png)
params=/imagecompressor=CTSquish /streaming=0
lockSource=false
server=skip
...

...[RC special images]
pattern=.*(_awesome)\.tif?
...
```

To add image processing presets

1. In a text editor, open the `AssetProcessorPlatformConfig.ini` file, located in the `lumberyard_version\dev` directory.
2. Specify the appropriate lines of code to define your new preset.
3. Save and close the file.

Using Image Naming Conventions

You can use any existing or created image processing presets (p. 280). To do this, append the suffix to the end of the file name before you add it to your Lumberyard directory.

Example

If you create an image that you want to use as a decal, add the _decal suffix to the file name before the extension. For example, to convert scorchmark.tif as a decal, rename it to scorchmark_decal.tif. You then add it to your Lumberyard directory, so that Asset Processor automatically processes the file.

There are a number of existing presets that you can use. For a full list, see the rc.ini file in the lumberyard_version\dev\Bin64\vc141\rc directory.

Some of the existing presets include the following:

- _diff – Albedo
- _spec, _refl – Specular textures
- _ddn – Normal map texture
- _ddna – Normal map texture with smoothness in alpha
- _bump – Converts grayscale texture to normal map
- _displ – Displacement map
- _decal – Decal
- _detail – Merged detail maps
- _cm, _cubemap – HDR reflection textures
- _cch – Color chart
- _mask – Grayscale mask
- _sss, _trans, _opac – Opacity

Using the Resource Compiler Image Tool

Before you can use the Resource Compiler image tool, you must install RC Shell Commands.

Note
You can also use the Texture Settings Editor to manage your texture settings for image files. For more information, see Texture Settings Editor (p. 1734).

To install RC Shell Commands

1. Open Lumberyard Setup Assistant (p. 16).
2. On the Install plugins page, click Install.
Once you install the RC Shell Commands, you can use the Resource Compiler image tool.

**To open the Resource Compiler image tool**

- Right-click an image file from within a file explorer and choose **RC Open Image**.

![RC Open Image](image)

The Resource Compiler image tool lists image processing presets defined in the `rc.ini` file, located in the `lumberyard_version\dev\Bin64vc141\rc` directory. Select a preset to create the `imagefilename.exportsettings` file that you can check in to your source control.

**To select an image processing preset and save it to an .exportsettings file**

1. In the Resource Compiler image tool, under **Preset**, select a preset from the drop-down menu.
2. (Optional) To see more information about the preset settings, click **Show preset info**.
3. Select other options you want to change, such as **MIP Control (p. 286)**.
4. Click **Generate Output**.

Your `imagefilename.exportsettings` file and the resulting `imagefilename.dds` output is saved in the same directory as your source image.
Generating Mip Maps

Mip maps are a sequence of optimized images made from one image. Each image is lower in resolution than the previous image by a power of two. Mip maps reduce the time and processing power it takes to render an image in a game. Lower resolution mip maps are used when the viewing distance is great enough that the loss of detail is not noticeable. Higher resolution mip maps are used when an object is close to the camera and needs to be displayed in detail.

When you enable mip maps in Lumberyard, Resource Compiler automatically generates six mip maps for that image. The largest mip map is the original size of your image, and each progressive mip map is smaller by a power of two.

Example

If your original image is 1024x1024, then you will have mip maps that are 1024x1024, 512x512, 256x256, 128x128, 64x64, and 32x32.
Example

If your original image is 4096x4096, then you will have mip maps that are 4096x4096, 2048x2048, 1024x1024, 512x512, 256x256, and 128x128.

Note

For most of the existing presets, mip maps are enabled by default. If you create your own presets, you must enable mip maps.

If mip maps are not enabled, then on the MIP Control tab, under Mip maps, none (0) appears.
To enable mip maps for an image processing preset

1. In a text editor, open the rc.ini file, located in the lumberyard_version\dev\Bin64vc141\rc directory.
For more information, see Creating Image Processing Presets (p. 280).

2. Add `mipmaps=1` to the definition.

**Example**

```xml
...[
    AwesomeImage]
    pixelformat=BC1
    pixelformat:es3=ETC2
    pixelformat:ios=PVRTC2
    ;pixelformat:es3=ASTC_4x4
    ;pixelformat:ios=ASTC_6x6
    rgbweights=ciexyz
    powof2=1
    mipmaps=1
    colorspace=sRGB,auto
    ;discardalpha=1
    filemasks=*_awesome*
...```

To disable mip maps, specify `mipmaps=0`.

**Adjusting Alpha Test on Mip Maps**

You can adjust the Alpha Test value for each mip map in the MIP Control tab of the Resource Compiler image tool.

**To adjust Alpha Test for each mip map**

1. For each mip map, adjust the sliders or enter a number in the box.
2. Select **Maintain alphatest coverage** to maintain shape and opacity from a distance when working with certain types of objects, such as leaves on a tree. If you don't select this option, the tree leaves may lose their silhouette in the lower resolution mip maps.
Configuring Image Processing

### MIP Control

- **Mip maps:**
  - Max, tiled (1)
- **AlphaTest adjustment:**
  - Mip1: 50
  - Mip2: 50
  - Mip3: 50
  - Mip4: 50
  - Mip5: 50
  - Mip6: 50
- **Filter scale:** 0

### Preset (defined in rc.ini)

- **List all presets:**
  - AlbedoWithGenericAlpha
- **Show preset info...**

### Preview

- **On**: Toggle preview mode.
- **Zoom In**, **Zoom Out**: Adjust zoom level.
- **Tiled**, **Bilinear**: Select preview mode.
- **RGB**: Normal RGB preview mode (no gamma correction).

### Resolution

- **1024x1024 Fmt:A8R8G8B8 Alpha:8**
- **Mem:4096.0kB**
- **1024x1024 Mips:9 Fmt:BC3 Mdls:RGB Alpha:32f8**
- **Mem:1365.3kB reduce:0 Flags:00000800 08x**

---

Version 1.28

290
Live Reloading and VFS

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

On the PC platform, live reloading does not require virtual file system (VFS), since the PC that is running the game is presumably also running the Asset Processor.

On non-PC platforms, VFS is required for live reloading to work, because otherwise assets would need to be deployed onto the game device as part of live reloading, incurring platform-specific costs and different asset pipelines. VFS enables the same behavior across all platforms using the same workflow. For debugging purposes, you can also enable VFS on a PC and point it at a remote Asset Processor to serve assets.

To enable VFS, you use the `bootstrap.cfg` configuration file.

The game runtimes and all tools can communicate with the Asset Processor through simple interfaces. Communication involves the following:

- Notification when assets are built and change, so as to reload them if possible.
- Request an immediate compilation of an asset, blocking until processing has completed.
- Request asset status, blocking until the status is known.
- Query the location of an asset source file, given an asset ID.
- Query the destination asset ID, given an asset source file name and path.

Not all asset types can live reload. If you are developing new asset types, keep the following guidelines in mind:

- When an asset loads, be prepared to substitute it for a temporary asset while it is compiling.
- If an asset is missing, query the status of the asset from the Asset Processor. This can determine whether the asset really is missing or whether it is in the queue for processing. Querying also moves the asset to the front of the queue for processing.
- If your asset is essential and it cannot live reload, use the blocking synchronous asset build request to make it build immediately. This moves the asset to the front of the queue and prevents the call from returning until the asset is compiled.
- Do not discard the original requested name when an asset is missing.
- Connect to the notification bus to learn when assets change and reload them when that happens.

Compiling Shaders for Release Builds

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you build game projects with Lumberyard, shaders for release builds should be compiled and packaged into .pak files.

**Mobile devices** – On mobile devices, runtime shader compilation is not supported for release builds. Shaders will compile at run time only if you are running in profile mode or debug mode and can connect to a remote shader compiler server (p. 1645). For more information, see the following:
• Connect to the Shader Compiler (p. 3176)
• Creating a Release App (p. 3204)
• Running the Shader Compiler on Amazon EC2 (p. 3239)

**macOS** – On macOS, the shader compilation pipeline depends on the High-Level Shader Language (HLSL) optimizer. You must connect to a shader compiler on your PC or macOS when running a game on macOS during development. This compiles the required subset of shaders for your game, on demand. For more information, see Building Shaders for macOS Games (p. 3265).

**Windows DirectX** – On Windows builds that use the DirectX module, runtime shader compilation is supported for release builds. However, it is highly recommended that you compile shaders into .pak files for performance reasons. Compiling shaders at run time can cause unwanted frame rate fluctuations. In addition, objects that use shaders compiled at run time may fail to appear until the shaders are successfully compiled.

The following shader .pak files are required for release builds:

- **Shaders.pak** – Required only if you want to support runtime compilation. Source shaders are located in the `lumberyard_version\dev\Engine\Shaders\` directory.
- **ShaderCache.pak** – Compiled shaders of all possible combinations that Lumberyard uses.
- **ShaderCacheStartup.pak** – Compiled shaders that are used during start.

### Generating Shader .pak Files

To generate shader .pak files, use the following tools:

- **Shader Compiler** – The shader compiler server generates the ShaderList.txt file that contains the list of all shaders for your game. This server can run locally or on a remote PC.
  
  For more information, see Remote Shader Compiler (p. 1645).
- **ShaderCacheGen.exe** – File that populates the local shader cache folder with all the shaders contained in the ShaderList.txt file.
  
  For more information, see ShaderCache.pak File Generation (p. 293).
- **lmbr_pak_shaders.bat** – Batch file that generates the ShaderCache.pak files.
  
  For more information, see ShaderCache.pak File Generation (p. 293).

### Shader Compiler Proxy

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Some mobile devices may be connected via a USB TCP/IP tunnel and may not have direct network access to a shader compiler server. The shader compiler proxy component in Lumberyard allows such devices to forward shader compiler requests through the Asset Processor connection.

This proxy connection only works for connecting to the shader compiler server on that protocol. It is not a general purpose network bridge or tunnel. To use the shader compiler proxy, open the `system_assetsplatform.cfg` file and modify the following values:
Shader Cache and Generation

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This section discusses the shader cache and how to generate shader cache .pak files.

Shader Cache

The shader cache stores a collection of parsed and precompiled shaders. Since the shader code is written with multiple definitions, Lumberyard can generate an enormous number of different shaders. Compiling shaders on demand at run time is only possible on the PC. On-demand shader compiling can cause freezes during the game play and uses extra memory. To reduce this overhead, all required shader combinations for a game are parsed, compiled, and stored in the shader cache.

The shader cache generally refers to the following files:

- **Shaders.pak** – Contains the shader source files, which is everything inside the `lumberyard_version\dev\Engine\Shaders\` directory, excluding EngineAssets.
  
  **Note**
  
  The actual shader source code (*.cfi and *.cfx) can be removed from this file for the final released version, and is not required anymore when the binary shaders are valid and available.

- **ShadersBin.pak** – Contains the binary-parsed shader information of the shader source code.
- **ShaderCache.pak** – Contains compiled shaders for all possible combinations that have been submitted to the remote shader compiler.
- **ShaderCacheStartup.pak** – Contains a small subset of the shader cache that is used during game start. This file is loaded into memory for quicker start times, but is not required. This cache is often used to hold the minimum required set of shaders to show a loading screen so that the rest of the loading can occur.

ShaderCache.pak File Generation

Creating a ShaderCache.pak file consists of running the `lmbr_pak_shaders.bat` batch script, which in turn runs ShaderCacheGen.exe to ensure the local cache directory contains all the shaders that are listed in the ShaderList.txt file. The `lmbr_pak_shaders.bat` script packs the contents of the cache directory, creates a ShaderCache.zip file, and then renames the file to ShaderCache.pak.

You can find the ShaderList_platform.txt file either from the remote shader compiler server or from the Lumberyard Editor directory. This file contains the list of all shaders which ShaderCacheGen.exe uses to produce the shader combinations for your game.

When running Lumberyard Editor, individual shaders are created as you view them. As such, you do not strictly need a remote shader compiler server to test game release mode or test shader pack generation. You just need access to the ShaderList_platform.txt file that is created in the
Shader Cache and Generation

However, only the shaders you have viewed on your local computer while running Lumberyard Editor will be listed in the ShaderList_platform.txt file. For this reason, it is recommended that you use a remote shader compiler server if possible.

Note
During development time, when you run the game or Lumberyard Editor, and before the shaders are packed into shader cache .pak files, loose shader files are created in the following directory: lumberyard_version\Dev\Cache\your_game\platform\user\cache

The following sections detail the steps used to generate ShaderCache.pak files:

ShaderCacheGen.exe

Lumberyard ships with ShaderCacheGen.exe, which is located in the lumberyard_version\dev\Bin64 directory. For macOS, the file is located in the BinMac64 directory. ShaderCacheGen.exe is a stripped-down version of the Lumberyard game launcher without the render viewport, and is used to populate the local shader cache directory with all the shaders contained in the ShaderList.txt file.

Running the ShaderCacheGen.exe will load the ShaderCacheGen.cfg file. This configuration file includes the IP address and other settings for the remote shader compiler that generates the shaders. The ShaderCacheGen.exe must connect to the remote shader compiler that's specified in the configuration file. The ShaderCacheGen.exe will not use the IP address and other settings that are specified in the system configuration files.

If you customized Lumberyard in any way, you must build Lumberyard and your game using the all profile. This argument builds both ShaderCacheGen.exe (and ensures that it is up-to-date) and the game .dll files that are required.

Enter the following command for your version of Visual Studio.

```
lmbr_waf build_win_x64_vs2017_profile -p all --targets=CrySCompileServer
```

If you don't want to (or can't) build using the all profile, you can alternatively just build the game_and_engine spec and the shadercachegen spec.

Enter the following command for your version of Visual Studio.

```
lmbr_waf build_win_x64_vs2017_profile -p game_and_engine
lmbr_waf build_win_x64_vs2017_profile -p shadercachegen
```

Packing the Shader Cache as Part of the Release Build

The ShaderCache.pak files are generated for release builds as part of the build process. To generate the files, the ShaderCacheGen.exe must be connected to the Remote Shader Compiler. This must be the same Remote Shader Compiler that you used to generate the shaders for your game. As long as this instance of the shader compiler is running, the release build process can generate the necessary .pak files for your release build. If the shader compiler isn't running or reachable for some reason, the release build process fails.

Packing the Shader Cache Using a Batch File

The lmbr_pak_shaders.bat file generates the ShaderCache.pak files, which are saved to the lumberyard_version\dev\build\platform\your_game directory. The batch file first calls the ShaderCacheGen.exe and then calls Tools\PakShaders\pak_shaders.py.

From a command line, navigate to the lumberyard_version\dev directory, and run the lmbr_pak_shaders.bat, and specify the location to the ShaderList_platform.txt file.
Example

\[F:\{lumberyard_version}\\dev\lmb\pak\shaders.bat game_project_name D3D11 pc C:\shader_compiler_server\ShaderList_DX11.txt]\]

Once the shader .pak files are created, you can move them as needed. For example, if you already built a release version of your game, you can place them with the rest of the .pak files.

**Packing the Shader Cache Manually**

If you want to use more complex build pipelines, you can pack the shader cache manually.

**To pack the shader cache manually**

1. Run ShaderCacheGen.exe to generate the shader cache so you can pack it later.
2. Zip all the shaders into a ShaderCache.zip file, then rename the file to ShaderCache.pak.

Each platform has different .pak files. See the following directory mapping for the PC platforms:

The PC platform should copy data from the following directory:

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\D3D11\] into the following destination directory:

\[shaders\cache\D3D11\]

ShaderCache.pak should contain everything from the previously listed subfolders.

ShaderBin.pak should contain only the *.cfxb and *.cfib files.

ShaderCacheStartup.pak should contain the following files:

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\lookupdata.bin -> Shadercache\\{platform\}\lookupdata.bin\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\CGPShader\FixedPipelineEmu* -> Shadercache\\{platform\}\CGPShader\FixedPipelineEmu*\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\CGPShader\Scaleform* -> Shadercache\\{platform\}\CGPShader\Scaleform*\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\CGPShader\Stereo* -> Shadercache\\{platform\}\CGPShader\Stereo*\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\CGVShader\FixedPipelineEmu* -> Shadercache\\{platform\}\CGVShader\FixedPipelineEmu*\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\CGVShader\Scaleform* -> Shadercache\\{platform\}\CGVShader\Scaleform*\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\CGVShader\Stereo* -> Shadercache\\{platform\}\CGVShader\Stereo*\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\lookupdata.bin -> Shadercache\\{platform\}\lookupdata.bin\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\Common.cfib -> Shadercache\\{platform\}\Common.cfib\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\fallback.cfxb -> Shadercache\\{platform\}\fallback.cfxb\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\fixedpipelineemu.cfxb -> Shadercache\\{platform\}\fixedpipelineemu.cfxb\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\FXConstantDefs.cfib -> Shadercache\\{platform\}\FXConstantDefs.cfib\]

\[\{lumberyard_version\}\\dev\cache\your_game\platform\user\shaders\cache\\{platform\}\FXSamplerDefs.cfib -> Shadercache\\{platform\}\FXSamplerDefs.cfib\]
Build Platforms

You can find the build platform subfolders listed in the following table in the `lumberyard_version\dev\Cache\your_game\platform\user\cache\shaders` directory.

<table>
<thead>
<tr>
<th>Build Platform</th>
<th>Build Platform Subfolder</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC, DirectX 11</td>
<td><code>\D3D11</code></td>
</tr>
</tbody>
</table>

Game Startup Sequence

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Compiled Lumberyard games start up in the following sequence:

1. The game reads the `bootstrap.cfg` file, which must contain the following information at a minimum:
   - Name of the game, and optionally, the name of the game DLL, if it differs from the game name.
   - Whether or not to connect to the Asset Processor on startup or listen for an incoming connection instead.
   - Whether or not to wait for an established connection before proceeding.
   - Whether or not to enable the virtual file system (VFS), which allows you to read assets remotely from a connected computer instead of having to deploy them to the game device. This also is required for live reloading to function on non-PC operating systems.
   - Which kind of assets to load. For example, you could configure the Android runtime to load `es3` assets, or `pc` assets, or `metal` assets. This determines which directory the game looks in for the assets so that the appropriate directory is also used for VFS.

2. The `lyconfig_default.xml` file is read.

3. VFS is started and enabled. All file access then goes through the VFS system. Besides the `bootstrap.cfg` file, executable files, DLL files, and associated OS files, nothing else needs to be deployed to the device. Instead, they can all be accessed remotely.

4. The `system_game OS_assets.cfg` file is read, where `assets` are the assets specified in the `bootstrap.cfg` file.

Asset Browser

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The **Asset Browser** displays all project assets in a source folder and file view to enable quick access and interaction. You can use the **Asset Browser** with other editor components to improve your development workflow, such as the viewport and the **Entity Inspector**.

The **Asset Browser** shows all files inside your watch folders that are specified in the `lumberyard_version\dev\AssetProcessorPlatformConfig.ini` file. In the **Asset Browser**, usable files appear white. Non-usable files such as `.exe` or `.zip` appear gray.

**To open the Asset Browser**

1. From Lumberyard Editor, choose **Tools, Asset Browser**.

   ![Asset Browser](image)

   To open the Asset Browser

2. Dock the **Asset Browser** window in Lumberyard Editor as needed.

   The **Asset Browser** displays source assets along with their products. For example, an `.fbx` file appears with its meshes and animations. However, if the source and product asset have the same name and extension, then the products are hidden to save space.

**Example**

The `WombearActor.fbx` file has two products, an actor and mesh.

![Example](image)

Files that are unprocessed or not usable in Lumberyard Editor appear gray.
With the **Asset Browser**, you can do the following:

**Contents**
- Drag and Drop (p. 298)
- Context Menu Operations (p. 300)
- Finding a Slice in the Asset Browser from the Viewport (p. 302)
- Filtering (p. 302)

**Drag and Drop**

You can drag assets from the **Asset Browser** into the viewport or the **Entity Inspector**.

When you drag an asset from the **Asset Browser** into the viewport, Lumberyard Editor does the following:

- Creates a new entity at the cursor's location.
- Adds the associated component, which is indicated by the icon next to the asset in the **Asset Browser**.
- Assigns the asset for that component's property.

**Example**

You can drag the `brokenrobot05.cgf` file to the viewport, which creates a new entity, adds a **Mesh** (p. 684) component, and assigns the asset into the **Mesh asset** property in the **Entity Inspector**.
Example

You can select multiple entities in the **Entity Outliner** and then select and drag an asset into the **Entity Inspector**. This adds the **Mesh** component and the associated mesh asset to the selected entities.

You can also drag the appropriate file type into a component property's field.

**Example**

You can drag an asset file (.cfg) into the **Material asset** field on the **Mesh** component. The boulder in the viewport displays the new asset.
Context Menu Operations

In the Asset Browser, right-click an asset to display a context menu. The following menu options appear:

- **Open with associated application** – Opens the file with the default application that you specify. For example, if you specified Photoshop to open .tiff files, this opens the file in Photoshop.
- **Open in Explorer** – Opens the asset in Windows Explorer.
- **Copy Name To Clipboard** – Copies the asset name.
- **Copy Path To Clipboard** – Copies the path of the asset.
- **Edit Texture Settings** – Opens the asset in the Texture Settings Editor. For more information, see Texture Settings Editor (p. 1734).

**Note**
You can also open some files in their respective Lumberyard tool, such as the **Lua Editor** for .lua files.

**FBX Files**

Some files, such as .fbx files, have an additional **Edit Settings** option, which opens the **FBX Settings** tool. For more information, see Customize FBX asset export with FBX Settings (p. 409).
Slices

The following additional options appear for slices:

- **Set Dynamic Slice** – Create a runtime version of a slice. For more information, see Working with Dynamic Slices (p. 524).
- **Unset Dynamic Slice** – Remove a dynamic slice.
- **Open in Slice Relationship View** – Show the dependencies and dependent members of the slice in a hierarchical view.

**Example**

The following image shows the Slice Relationship View for the Maze_Decor_Interior slice from the Starter Game. For more information, see Working with Slices (p. 510).

**Source Control**

If source control is enabled, the following additional options appear:

- **Check Out** – Check out the file in source control.
- **Undo Check Out** – Undo check out for the file.
• **Get Latest Version** – Get the latest version of the file.
• **Add To Source Control** – Add the file to source control.

### Finding a Slice in the Asset Browser from the Viewport

In the viewport, right-click the slice or slice entity and choose **Find slice in Asset Browser**. The Asset Browser navigates to the corresponding slice.

### Filtering

You can filter what appears in the Asset Browser by file name, asset type, or both.

**To filter your assets**

1. In the Asset Browser, choose the filter icon in the upper right.
2. Select one or more asset types.
3. To filter by file name, type search terms into the search bar. The filtering process is dynamic, which means that search results update as you type.

4. To reset your selection, in the Filter by drop-down menu, click Reset or Clear.

The search function also has the following features:

- Recognizes multiple words and performs an and search on assets and directories that contain all of the search terms in any order.
- Is not case sensitive.
- Displays product assets that match the search string.
- Displays source assets that match the search string and contain at least one valid product.
- Displays directories that match the search string and contain at least one valid product. All products in each displayed directory and its subdirectories are displayed.
Programming the Lumberyard AZCore Runtime Asset System

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Lumberyard Editor and Lumberyard runtime code use the AZCore runtime asset system to asynchronously stream and activate assets. This topic describes the workflow of the classes in the asset system and shows how to load already-built assets into a running instance of the engine.

Note
For information on compiling and building assets, see Working with the Asset Pipeline and asset files (p. 247).

Asset System Classes

The Lumberyard asset system includes the following classes and class families:

- AZ::Data::AssetData Derived Classes (p. 304)
- AZ::Data::AssetManager (p. 306)
- AzFramework::AssetCatalog (p. 307)
- AZ::Data::AssetHandler Derived Classes (p. 308)

The following sections describe these classes in detail. For the source code, see the lumberyard_version\dev\Code\Framework\AzCore\AzCore\Asset directory.

AZ::Data::AssetData Derived Classes

An AssetData class represents the data of an asset that is loaded in memory. To describe a particular kind of asset, derive from the AssetData base class. The base class provides an AssetID and a reference count member variable for the asset.

The following Lumberyard classes derive from AssetData:

<table>
<thead>
<tr>
<th>AssetData Class</th>
<th>Source Code Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptAsset</td>
<td>lumberyard_version\dev\Code\Framework\NoteBeg\AzCore\AzCore\Script\ScriptAsset.h</td>
</tr>
<tr>
<td>SliceAsset</td>
<td>lumberyard_version\dev\Code\Framework\AzCore\AzCore\Slice\SliceAsset.h</td>
</tr>
<tr>
<td>MeshAsset</td>
<td>lumberyard_version\dev\Gems\LmbrCentral\Code\include\LmbrCentral\Rendering\MeshAsset.h</td>
</tr>
<tr>
<td>ParticleAsset</td>
<td>lumberyard_version\dev\Gems\LmbrCentral\Code\include\LmbrCentral\Rendering\ParticleAsset.h</td>
</tr>
</tbody>
</table>

Note
Adding your own asset type to Lumberyard includes the following high-level steps:
1. Derive your type from `AssetData`.
2. Declare an `AZ_RTTI` type for the asset to ensure that it has a UUID.
3. Add the member fields or structs that store your data in memory at run time.

For more information, see Adding an Asset Type to Lumberyard (p. 308).

**AZ::Data::Asset<T> Templated Class**

Typically, components which use assets directly or indirectly do not have a pointer to your `AssetData`-derived class; instead, they have a member of type `Asset<T>`. The `AZ::Data::Asset<T>` templated class is a wrapper that is similar to a smart pointer, and the `T` templated type is an `AssetData`-derived class.

The use of `Asset<T>` provides the following benefits:

- Automatic dependency tracking for components that are part of slices.
- Automatic reference counting.
- Automatic reloading of asset data when the asset changes on disk.
- Explicit lifecycle management functions like `IsLoaded()` and `QueueLoad()`.
- Reference count tracking to ensure correct behavior for copy operators.
- The ability to control how the `Asset<T>` class loads data. To specify how the asset loads, you pass flags to the constructor of the `Asset<T>` member variable.

The following options are possible:

- The class automatically starts loading its data. The class waits for the data to be ready before it activates the component for which the data is intended.
- The class queues the load of your asset data.
- The class waits for you to load the data explicitly.

**Note**

A loaded asset remains loaded as long as an active `Asset<T>` points to it. The asset manager does not reference count the asset. The asset is unloaded when the last system with a reference to the `Asset<T>` drops its reference and the reference count on the asset goes to 0.

**Integration with UI Property Grids**

The `Asset<T>` member fields of your component can appear in UI property grids like those in the Entity Inspector. To make a component's field available in Lumberyard Editor, make the `Asset<T>` field a member variable and reflect it into the editor. When you do so, game developers can drag an asset from the Asset Browser onto the property field to assign the asset to the component.

Note the following points:

- Reflect the `Asset<T>` member variable just as you reflect other member variables of your component.
- Lumberyard handles asset IDs for you automatically. You do not have to handle them explicitly.
- `Asset<T>` fields serialize the `AssetId` and other information such as the last known name of that `AssetId`.
- After an `AssetId` is assigned to a component, the `AssetId` is saved when the component is saved. The next time the component loads, the asset is automatically loaded if you specified the appropriate flag in the `Asset<T>` constructor.
AZ::Data::AssetManager

AZ::Data::AssetManager is the central hub for retrieving assets. If you configure the Asset<T> fields of a component to load their assets automatically, you do not need to communicate directly with the asset manager. The AssetManager class performs the following tasks:

- Maintains a hash table that maps asset IDs to the instances of Asset<T> that are currently loaded.
- Calls FindAsset to see if an asset is already loaded. If the asset is not currently loaded, FindAsset returns a null reference.
- Automatically reloads assets as they change on disk.
- Notifies listeners about asset lifecycle changes. Events like asset loading or unloading are signalled on the AssetBus. The callback-based adapter for this bus is called AssetBusCallbacks. For more information, see the AssetCommon.h file.

To get an asset, call GetAsset. If the reference count is greater than zero, GetAsset returns an Asset<T> that is already loaded. If no Asset<T> is currently loaded, GetAsset starts loading a new instance of Asset<T>.

Example: Loading an Asset Using Asset Manager

The following code example uses AssetManager to load a script asset.

```cpp
m_scriptAsset = 
AZ::Data::AssetManager::Instance().GetAsset<AZ::ScriptAsset>(assetIdToLoad);
AZ::Data::AssetBus::Handler::BusConnect(m_scriptAsset.GetId());
```

In the example, m_scriptAsset is a field of type Asset<ScriptAsset>.

For related code, see lumberyard_version\dev\Code\Framework\AzToolsFramework\AzToolsFramework\ToolsComponents\ScriptEditorComponent.cpp.

Note the following points:

- GetAsset loads the asset asynchronously. By assigning the asset to the member m_scriptAsset, you ensure that the reference count is at least 1.
- The code connects to the AssetBus to receive notifications when the script asset is loaded or becomes ready.
- If the asset is already loaded, the AssetBus delivers the OnAssetReady event as soon as the connection is made to the bus. Because the connection to the bus triggers a callback about the asset's state, you do not have to write code to check the state.

More About Automatic Reloading

An asset can change on disk after the asset has been loaded (and therefore has a reference count greater than zero). When this occurs, the asset manager creates an instance of the updated asset and loads it in the background. When the updated asset is finished loading, two Asset<T>'s temporarily exist. One Asset<T> points to the old AssetData instance in memory, and one to the new. Both instances have the same AssetId. However, now when you request the asset by AssetId, the asset manager returns the new instance and increases the reference count of the new instance. The asset manager also sends the OnAssetReloaded(Asset<T>) event to the AssetBus. This notifies other systems to reload the asset by replacing their current member Asset<T> with the new instance. It also keeps the reference count from reaching zero for the duration of the callback.

The following code shows a component that has a member variable of type Asset<T> that handles live reloading. First, the component connects to the bus to monitor for asset reloading events.
When the AssetManager notifies that a new script has been reloaded, the code for the
OnAssetReloaded method cleans up old pointers. The code also assigns the asset, which updates the
reference count for the new and old versions of the asset.

```cpp
void ScriptComponent::OnAssetReloaded(AZ::Data::Asset<AZ::Data::AssetData> asset)
{
    // Clean up any pointers to the old AssetData.
    UnloadScript();
    m_script = asset; // This assignment increments the reference count of the new asset.
    // The old asset reference count decrements.
    // Re-establish state into the new AssetData.
    LoadScript();
}
```

Because `m_script` is of type `Asset<ScriptAsset>`, it can simply use `Asset<T>`'s `operator=` to drop
the reference count on the old `Asset<T>` and replace it with the new `Asset<T>`.

This way of handling automatic reloading gives components the flexibility to decide how to deal with
changes to assets. For example, components might choose among the following options:

1. Save the new `Asset<T>` on a queue and for later processing.
2. Discard the new `Asset<T>` and keep the old data.
3. Swap the references to the old and new versions immediately, as the script component does.

Note the following points:

- Because `Asset<T>` instances are reference counted, the internal `AssetData` object that they wrap is
  not deleted until all classes that have a reference to it clear that reference.
- If `OnAssetReloaded` is called and the code does not store the new `Asset<T>`, the reference count
  becomes zero and the asset is unloaded. Existing `Asset<T>` instances that point at the old data
  remain valid until they are dropped.
- Because messages like `OnAssetReloaded` are always delivered in the main thread, mutexes are not
  required.

**AzFramework::AssetCatalog**

The asset catalog is a set of lookup tables that notifies the Lumberyard asset system when assets on
the file system change. The asset manager monitors the `AssetCatalogEventBus`. When the bus
delivers the `OnCatalogAssetChanged` event, the asset manager starts upgrading assets. This is how
live reloading is implemented.

To receive notifications about assets that change on disk, connect to the `AssetCatalogEventBus`. Then
use the `AssetCatalogRequestBus` to make requests to the `AssetCatalog` to resolve assets by ID. For details, see the `AssetManagerBus.h` file.

The `AssetCatalogRequestBus` contains other functions that look up asset dependencies, enumerate
assets, and perform other low-level tasks. In most cases you do not have to use these functions directly.

**Note**

You do not have to use the asset catalog directly unless you write low-level code that performs
custom file processing. If you use the higher level systems like `Asset<T>`, `AssetData`, and
`AssetManager`, these classes communicate with the catalog for you.
To look up asset file information manually, you can pass an AssetId to the AssetCatalog. AssetCatalog returns a struct that contains the file's type, size, canonical name, and location.

**AZ::Data::AssetHandler Derived Classes**

When you create a new type of asset (p. 308), you also create an AssetHandler for the new asset type. The role of the asset handler is to create, load, save, and destroy assets when the asset manager requests it. After your asset handler creates an empty instance of your asset type, it loads serialized data into the in-memory representation of AssetData.

To create a handler for a specific asset type, derive from the AssetHandler class and register an instance of the handler with the asset manager. Because asset handling functions can be called from multiple threads, the handlers must be thread-safe. The handler can block the calling thread while the asset is loading.

**Asset System Workflow**

Lumberyard loads assets in the following two ways:

- **Implicit** – When classes and structs contain Asset<T> members. When a structure deserializes, the serialization system checks whether the structure contains a member of type Asset<T>. If so, the serialization system calls GetAsset() to retrieve the asset from AssetManager.
- **Explicit** – When AssetManager::GetAsset() or Asset<T>::QueueLoad is called explicitly.

The following steps summarize the workflow of the asset system.

1. GetAsset(assetId) calls AssetManager implicitly (through the serialization system) or explicitly.
2. AssetManager calls GetAssetInfoById to retrieve the information about the asset file.
3. If the asset is already loaded in the m_assets asset map, AssetManager returns a new Asset<T> instance of the existing asset and increments the reference count.
4. If the asset is not already loaded, AssetManager uses the information returned by GetAssetInfoById to look up the AssetHandler for the asset type.
   a. AssetManager calls the asset handler's CreateAsset function to create a new empty instance for the data.
   b. AssetManager inserts the asset into the empty instance.
   c. AssetManager creates a loading job in a job worker thread pool. To load the asset data, the thread pool calls the loadAssetData function on the handler in the worker pool.
   d. If the AssetManager was told to block while the asset loads, it stops processing until the asset is loaded.
   e. AssetManager returns the Asset<T> member.

**Conclusion**

While AssetCatalog, AssetHandler, and AssetData are part of the asset system, consumers of an asset deal only with Asset<T> and AssetManager.

**Adding an Asset Type to Lumberyard**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
When you develop a game, you might need to add a new kind of asset to Lumberyard. The new asset could be a configuration file, a game-specific data asset, or structured data for which you created an editor. This topic guides you through the process of adding a custom asset type to Lumberyard.

For an overview of the Lumberyard asset system, see Programming the Lumberyard AZCore Runtime Asset System (p. 304).

Topics
- Overview (p. 309)
- A. Registering Your Asset with the Asset Pipeline (p. 310)
- B. Enabling the Engine to Load and Stream the Asset (p. 312)
- C. Customizing UI Interaction (p. 314)

Overview
At a high level, adding an asset type to Lumberyard involves three steps:

A. Registering the asset with the asset pipeline.
B. Enabling the Lumberyard engine to load and stream the asset.
C. Optionally customizing your asset's interaction with the Lumberyard Editor UI and the Asset Browser.

After this overview, each step is covered in more detail. Not all of the steps are required. The steps are arranged in order of increasing integration with the Lumberyard asset pipeline and editor.

A. Register the Asset with the Asset Pipeline
You register your asset with the asset pipeline for common operations like building, processing, copying, and deploying. This registration assigns an appropriate type ID to the asset type and enables the asset to be added to the cache and the asset database. Depending on your asset file type, you can register it by creating a copy rule or writing code.

Registering by Copying
You can deploy your asset file by copying if the following are true:

- The processed or compiled version of your asset is the same across all operating systems and devices.
- The asset declares no dependencies on other assets.
- The asset doesn't need to be rebuilt when other source files change.

Registering by Creating a Builder
If your asset requires processing, compiling, or optimizing at run time, you can do one of the following:

- Create a BuilderSDK builder (p. 318) to transform your source asset into its compiled form.
- If your asset is extracted from a scene file like FBX and you want to integrate with its corresponding pipeline, create a scene pipeline plug-in.

Copying Versus Creating a Builder
It might be inefficient to use your source assets in your game under the following circumstances:

- You have extraneous tool data inside your asset that is not necessary to ship for each operating system or device.
- You want to do verify or error check your files.
• You want to optimize your data.

In these cases, consider writing a BuilderSDK builder (p. 318) instead of just copying your raw assets, even if it only compresses or converts the assets into binary format.

B. Enable the Engine to Load and Stream the Asset

After you register your asset with the asset pipeline, you must enable the Lumberyard engine to load your asset type at run time and stream it asynchronously.

Enabling your asset for loading and streaming involves the following steps:

1. **Create an asset data type** – Create a class derived from `AZ::Data::AssetData` that represents your loaded asset data. This class must be registered with `AZ_RTTI`. The UUID of the class is also the UUID of your asset in the Asset Processor database.

2. **Create an asset handler** – Create a singleton class that derives from `AZ::Data::AssetHandler` that loads, saves, and reads your data from buffer.

3. **Register your asset type handler with the asset system** – To do this, implement a `RegisterHandler` function inside your handler.

4. **Use your custom asset** – Inside components, use `Asset<T>` members, which are reflected to the editor and for serialization, to refer to your asset.

C. Customize UI Interaction

Optionally, you can describe your asset to the UI (and other systems) and customize your asset's interaction with it, including with the **Asset Browser**.

1. **Describe your asset type to the UI** – Use the `AssetTypeInfoBus` to provide integration with UI features like the **Asset Browser**.

2. **Customize drag-and-drop behavior** – If you have components, you can add reflection to them to enable custom behavior. For example, when a game developer drags your asset to the viewport, you can make your component be the default component that is spawned.

3. **Customize the context menu** – You can add context menu actions to interact with your asset.

4. **Customize file open behavior** – Customize file open operations to create custom behaviors when interaction occurs with your asset. For example, you can make your asset type open in a custom editor or create custom entities.

In addition, you can use the thumbnail API to generate thumbnails for your asset for use in the **Asset Browser** and other UI contexts. Although not described here, the related source code is located in the `lumberyard_version\dev\Code\Framework\AzToolsFramework\AzToolsFramework\Thumbnails` directory.

A. Registering Your Asset with the Asset Pipeline

When you register your asset with the asset pipeline, you define your asset to the asset system and asset processor. Depending on your asset file type, you can register it by creating a copy rule or by writing a BuilderSDK builder (p. 318).

Copying Assets

To copy assets, you typically create a copy rule in a configuration file. The configuration file that you use depends on your implementation goals.

• To copy assets as-is into the cache, add a copy rule to your `lumberyard_version\dev\AssetProcessorPlatformConfig.ini` file.
If the asset type is specific to a gem, add it to your gem's root directory (the directory with the gem.json file) instead of to the AssetProcessorGemConfig.ini. This overrides the platform configuration file but is overridden by the game configuration file.

If the asset type is specific to the game project, add it to your game directory's AssetProcessorGamePlatformConfig.ini file instead of to the AssetProcessorPlatformConfig.ini file.

A copied asset uses a copy rule entry similar to the following example. Rule names must be unique. In the example, the *dba copy rule copies *.dba files and uses a UUID to register their type. The asset shows in the catalog as the type that you designate. The type maps the asset to the AssetHandler that you create. This enables the handler to load the asset.

```
[RC dba]
glob=*.dba
params=copy
productAssetType={511562BE-65A5-4538-A5F1-AC685366243E}
version=1
```

For more information, see Configuring the Asset Pipeline (p. 269).

**Custom-Built Assets**

For custom-built assets, you can write a BuilderSDK builder or a Scene API plug-in. If your asset is extracted from an .fbx file, you can write a Scene API plug-in. With either approach, you do not need to edit .ini files as you do when copying assets.

**Writing a BuilderSDK Builder**

When you write a BuilderSDK builder to create the asset, it should fill in the product info struct for each asset that it creates. The product info struct includes the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>The name of the asset file.</td>
</tr>
<tr>
<td>Asset type</td>
<td>The UUID of the class that you derived from Az::AssetData.</td>
</tr>
<tr>
<td>SubID</td>
<td>Any u32 integer that disambiguates different outputs from the same source.</td>
</tr>
<tr>
<td>Legacy SubIDs</td>
<td>SubIDs for backward compatibility.</td>
</tr>
</tbody>
</table>

For more information and examples, see the Asset Builder API (p. 318).

**Writing a Scene API Plug-in**

The Scene API provides boilerplate code so that you only have to write a few code hooks to create a plug-in. For an example Scene API plug-in, see the SceneLoggingExample gem in the lumberyard_version\dev\Gems directory. The SceneLoggingExample gem shows how logging can be added to the asset pipeline. The gem adds a scene plug-in that outputs logs for each node in the file. The gem is a good starting point for your own code. You can replace the log outputs with the data collection and/or compiling functions that you want.

**Registration Versus Integration**

After you register the asset with the asset system, you can use the AssetCatalogRequestBus to find the asset in the catalog by its ID or other attributes. The result of the lookup gives you the asset's path, size, and other information. You can use standard file handling (p. 329) to load the asset.
While simple registration of an asset can be useful in some circumstances, full integration with the Lumberyard asset system offers many advantages, including the following:

- Automatic live reloading
- Automatic asynchronous streaming
- Automatic drag-and-drop support in the Asset Browser
- Thumbnails

### B. Enabling the Engine to Load and Stream the Asset

The following sections describe the steps required for integration with Lumberyard's asset system.

#### 1. Create an AssetData Type

To represent an image that is usable in-memory for your asset, derive a class from `AZ::Data::AssetData`. You do not load the asset from the `AssetData-derived class. Instead, the class actually represents your asset in memory, and the game uses it directly for processing. When you request the asset from `AssetManager`, the asset manager reference counts it, and the asset class is shared by any consumer of the asset. When you create an asset class, keep in mind the following:

- You can place `AssetData-derived classes in gems.
- Your asset class must have an AZ_RTTI declaration, including a UUID for the class's type. The UUID that you use for this type is actually the asset type that it represents. The UUID matches the type that you use in your asset builder or your copy rule.
- While an `AssetData-derived class typically contains asset data that it stores in memory and that you use at run time, this is not a requirement. If you have special requirements (like third-party considerations), your class can simply provide a handle to a foreign system that contains the asset data.
- For an example class that derives from `Az::Data::AssetData`, see the `ScriptCanvasAsset` class in the Lumberyard code at `lumberyard_version\dev\Gems\ScriptCanvas\Code\Editor\Include\ScriptCanvas\Assets\ScriptCanvasAsset.h`.

#### Asset Instances and Asset Data

It is important to understand the difference between asset instances and asset data.

An `AssetData-derived class contains the asset data that is shared among instances that use that data. Data that is specific to an instance should live in the instance itself. For example, an animation asset usually contains actual animation curves and samples. Only one such asset is loaded even if many characters use the animation. However, each character instance has an additional block of data that keeps track of the frame that the animation is on for the instance.

`AssetData objects are not immutable, and often change over time. For example, textures can load higher level of detail (LOD) objects, and procedural meshes or textures can change over time. However, all instances that use an asset get the same pointer to the asset and therefore change when the asset changes. This is also true when instances have a pointer to an asset in a foreign system.

#### 2. Create an Asset Handler

You write an asset handler to read assets from buffer and convert them into your `AssetData type. Only one such handler generally exists for each type of asset. The asset handler class is your asset factory for that type of asset because it can create and destroy your `AssetData-derived class. The `AssetHandler is a singleton instance whose functions are called in asynchronous job threads.

**Note**

You can place asset handlers in gems. For information on adding code to gems, see Using Gems to Add C++ Code to a Lumberyard Game (p. 1086).
Using the Generic Asset Handler for Structured Data

If your asset is a serialized `AZ::Reflected::ObjectStream`, you can use `GenericAssetHandler`, located in the `lumberyard_version\dev\Code\Framework\AzFramework\AzFramework\Asset\GenericAssetHandler.h` file. `GenericAssetHandler` calls the usual `Serialize` and `Deserialize` functions in its callbacks that load assets and registers your types for you.

Because assets like classes, lists, vectors, and properties are structured data, they are suitable for handling by `GenericAssetHandler`.

Writing a Handler for Custom Assets

If you have a custom asset (like bytes or custom formats), you must write your own `AssetHandler`. To do so, implement the `CreateAsset`, `LoadAssetData`, and other functions in the following code:

```cpp
// AZ::Data::AssetHandler
// CreateAsset is almost always implemented in the form:
// return aznew (your assetData class type)(id)
AZ::Data::AssetPtr CreateAsset(const AZ::Data::AssetId& id, const AZ::Data::AssetType& type) override;

// LoadAssetData is the main loading function. The stream that you are given is // already attached to the file and ready to read into the 'asset' variable.
bool LoadAssetData(const AZ::Data::Asset<AZ::Data::AssetData>& asset, AZ::IO::GenericStream* stream, const AZ::Data::AssetFilterCB& assetLoadFilterCB) override;

// The following functions are for legacy compatibility. Most implementations of the functions // simply wrap the file in a GenericStream and pass it to the LoadAssetData function above.
bool LoadAssetData(const AZ::Data::Asset<AZ::Data::AssetData>& asset, const char* assetPath, const AZ::Data::AssetFilterCB& assetLoadFilterCB) override;

// Destroy your asset in case you need to do cleanup. DestroyAsset usually just calls delete.
void DestroyAsset(AZ::Data::AssetPtr ptr) override;

// The AZ_RTTI type UUID(s) of your AssetData-derived classes.
void GetHandledAssetTypes(AZStd::vector<AZ::Data::AssetType>& assetTypes) override;

// CanHandleAsset usually returns true. Only useful if more than one handler is present for a single type.
bool CanHandleAsset(const AZ::Data::AssetId& id) const override;
```

After you create your handler, install the class that you derived from `AssetHandler`.

To install your asset handler with the asset system

1. Create an instance of the handler class in your component.
2. Call `RegisterHandler` on the asset manager instance. For an example, see the `lumberyard_version\dev\Code\Framework\AzCore\AzCore\Script\ScriptSystemComponent.*` files.

   **Note**
   Your component does not have to be a system component, but the component must exist to handle load request calls.

Example Asset Handlers

For example asset handlers, see `MeshAssetHandler`, `ParticleAssetHandler`, and `ScriptAssetHandler` in the following locations:
3. Register Your Asset Type and Handler with the Asset System

In this step, register your handler to handle a specific asset type, which is the UUID of your asset data.

Typically, you implement the registration code inside your handler in a `RegisterHandler` function. You call the `RegisterHandler` function during initialization, as in the following example. The example code registers an `<AssetType>` (that is, the UUID of the `AssetData` class type) with the asset catalog.

```cpp
AZ::Data::AssetCatalogRequestBus::Broadcast(&AZ::Data::AssetCatalogRequests::EnableCatalogForAsset, AZ::AzTypeInfo<AssetType>::Uuid());
AZ::Data::AssetCatalogRequestBus::Broadcast(&AZ::Data::AssetCatalogRequests::AddExtension, m_extension.c_str()); // The extension of your type.

// Register your handler.
AZ_Assert(AZ::Data::AssetManager::IsReady(), "AssetManager isn't ready!");
AZ::Data::AssetManager::Instance().RegisterHandler(this, AZ::AzTypeInfo<AssetType>::Uuid());
```

After you describe your asset type to the Lumberyard engine in this way, your custom asset becomes available in the Asset Browser for dragging and dropping. The asset displays its information and automatically updates.

4. Use Your Custom Asset in the Editor and Engine

At this point, you can use your custom assets in your structures and components.

To create components in the component editor

1. Add fields of type `AZ::Data::Asset<T>` to your components. To do so, use the following syntax. In the following example, `MyAsset` is the class that you derived from `AssetData`.

   ```cpp
   AZ::Data::Asset<MyAsset> m_myAsset;
   ```

2. Reflect the fields that you added by using editor reflection. For more information, see Reflecting a Component for Serialization and Editing (p. 977).

3. (Optional) In the constructor of your class, override the `m_myAsset` constructor to implement the serializer's behavior. For more information, see `m_script` in the `lumberyard_version\dev\Code\Framework\AzFramework\AzFramework\Script\scriptcomponent.cpp` file.

After you perform these steps, your component appears in the component editor. The component has a field to which you can drag and drop the asset from the Asset Browser. When the asset is dragged on the socket, the component accepts it. At this point, runtime or editor code can call functions like `QueueLoad` on the asset.

C. Customizing UI Interaction

(Optional) You can describe your asset type to the UI and then use EBus listeners (p. 1851) to customize your asset's interaction with the Asset Browser.
1. Describing Your Asset Type to the UI

In addition to deriving your asset handler from `Az::Data::AssetHandler`, you can derive it from `AZ::AssetTypeInfoBus::Handler` for user interface purposes. AssetTypeInfoBus tells Lumberyard the friendly name of your asset type, enables it to appear in the **Asset Browser**, gives it icons, and provides other UI–related features.

```cpp
// AZ::AssetTypeInfoBus::Handler
// Return the AZ_RTTI typeid of your AssetData.
AZ::Data::AssetType GetAssetType() const override;

// Get a friendly display name for the Asset Browser and GUI.
const char* GetAssetTypeDisplayName() const override;

// Get a friendly group name ("textures", "Meshes"...) for Asset Browser filtering.
const char* GetGroup() const override;

// Get the name of the icon image file to use in the Asset Browser.
// For example, "Editor\Icons\Components\StaticMesh.png".
const char* GetBrowserIcon() const override;

// Specify the AZ_RTTI type of an editor component for GetComponentTypeId.
// After doing so, dragging this asset type to the viewport will:
// 1) Spawn an entity with the component
// 2) Assign the asset to the component for you.
AZ::Uuid GetComponentTypeId() const override;

void GetAssetTypeExtensions(AZStd::vector<AZStd::string>& extensions) override;
```

Use EBus Listeners to Implement Custom Behavior

To customize how your asset type interacts with the **Asset Browser** and viewport, use EBus listeners to override or supplement behaviors like the following:

1. Default drag-and-drop behavior for viewport operations.
2. Default context menu (right-click) behavior for the **Asset Browser**.
3. Default file open behavior.

**Note**

Because a double-click is a request to open a file, when you override file open behavior, you also override double-click behavior.

2. Customizing Drag-and-Drop Behavior

To customize the default drag-and-drop behavior, you can edit the existing code in the `AzAssetBrowserRequestHandler.cpp` file. A more modular approach is to add a handler with higher priority to your gem.

To customize the default drag-and-drop behavior, create code that listens on the `DragAndDropEventsBus`, as in the following example.

```cpp
AzQtComponents::DragAndDropEventsBus::Handler::BusConnect(AzQtComponents::DragAndDropContexts::EditorViewport);
```

Your component now receives the DragEnter, DragMove, DragLeave, and Drop events shown in the following code.

```cpp
DragEnter(QDragEnterEvent* event, AzQtComponents::DragAndDropContextBase& context)
DragMove(QDragMoveEvent* event, AzQtComponents::DragAndDropContextBase& context)
```
DragLeave(QDragLeaveEvent* event)
Drop(QDropEvent* event, AzQtComponents::DragAndDropContextBase& context)

The events are described in the following table.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DragEnter</td>
<td>Called when your item is dragged over the viewport.</td>
</tr>
<tr>
<td>DragMove</td>
<td>Called repeatedly after your item accepts the event.</td>
</tr>
<tr>
<td>DragLeave</td>
<td>Called when the item leaves the viewport, provided DragEnter accepted the event earlier.</td>
</tr>
<tr>
<td>Drop</td>
<td>Called if the game developer drops the item into the viewport.</td>
</tr>
</tbody>
</table>

For more information, see lumberyard_version\dev\Code\Framework\AzQtComponents\AzQtComponents\Buses\DragAndDrop.h and lumberyard_version\dev\Code\Sandbox\Editor\AzAssetBrowser\AzAssetBrowserRequestHandler.h.

Priority of Handling

Bus handlers on the DragAndDropEventsBus are sorted by the return value of GetPriority(). Because the first handler to accept the event gets priority, you can override the default behavior by positioning your component before others.

When assets from the Asset Browser are dragged, the MimeData function in the drag event contains one or more AssetBrowserEntry pointers.

Several utility functions make it easier to implement your drag handlers. The following example is from lumberyard_version\dev\Code\Sandbox\Editor\AzAssetBrowser\AzAssetBrowserRequestHandler.cpp:

```cpp
AssetBrowserEntry::ForEachEntryInMimeData<SourceAssetBrowserEntry>(event->mimeData(),
    [&](const SourceAssetBrowserEntry* source) {
        SpawnEntityAtPoint(source, viewportDragContext, spawnedEntities, spawnTicket);
    });
```

ForEachEntryInMimeData<T> takes one of the types of AssetBrowser entries (SourceAssetBrowserEntry, ProductAssetBrowserEntry, FolderAssetBrowserEntry, RootAssetBrowserEntry) and calls your callback for each of the entries found. You can use these callbacks to detect events and react to them appropriately.

Checking Event Ownership

It is important to note that Asset Browser elements are not the only objects that go through this drag operations pipeline. Files and other objects of drag operations can produce drag events. For this reason, it is important to verify that a drag event that you received corresponds to your object before you take any action. Your code should not assume that your object is the one being dragged.

To perform the check, examine the event's isAccepted() Boolean property. If the event has already been accepted, do not take action. You can also check the status of the event by examining the mimeData attached in event->mimeData() to decide whether to accept it.

The following related code is from AzAssetBrowserRequestHandler.cpp:

```cpp
// If a listener with a higher priority has already claimed the event, do not touch it.
```
ViewportDragContext* viewportDragContext = azrtti_cast<ViewportDragContext*>(&context);
if ((!event) || (!event->mimeData()) || (event->isAccepted()) || (!viewportDragContext))
{
    return false;
}

The Lumberyard drag-and-drop system uses the same semantics as the Qt drag and drop system. For more information, see Drag and Drop in the Qt documentation.

3. Customizing the Context Menu

Whenever the context menu is requested, an EBus is invoked that listeners can use to register additional actions to it. To add items to the context menu, monitor the AssetBrowserInteractionNotificationsBus::Handler and implement AddContextMenuActions, as in the following example.

```cpp
// A notification that a context menu is about to be shown provides an opportunity to add actions to it.
virtual void AddContextMenuActions(QWidget* /*caller*/, QMenu* /*menu*/, const AZStd::vector<AssetBrowserEntry*>& /*entries*/) {};
```

4. Customizing File Open Behavior

Right or double-clicking an asset in the Asset Browser or elsewhere in the UI triggers the file open behavior. The default behavior passes the asset file to the operating system.

You can override this behavior by deriving from the AssetBrowserInteractionNotificationsBus::Handler and implementing AddSourceFileOpeners, as in the following code.

```cpp
virtual void AddSourceFileOpeners(const char* /*fullSourceFileName*/, const AZ::Uuid& /*sourceUUID*/, SourceFileOpenerList& /*openers*/) {};
```

A source file opener provides the Open with... feature in the UI. You can return multiple source file openers if you have more than one.

The following code shows how to add source file openers.

```cpp
void AssetBrowserContextProvider::AddSourceFileOpeners(const char* fullSourceFileName, const AZ::Uuid& sourceUUID, AzToolsFramework::AssetBrowser::SourceFileOpenerList& openers)
{
    using namespace AzToolsFramework;
    // Get the details of the source file based on its UUID.
    if (const SourceAssetBrowserEntry* source = SourceAssetBrowserEntry::GetSourceByAssetId(sourceUUID))
    {
        // Specify actions to take when the source file is not handled.
        if (!HandlesSource(source))
        {
            return;
        }
    }

    // Create as many openers as you want. For each opener, specify a unique identifier, an icon, and a Lambda function to call.
    openers.push_back({"Lumberyard_FBX_Settings_Edit", "Edit Settings...", QIcon(), [this](const char* fullSourceFileNameInCallback, const AZ::Uuid& sourceUUID*)
    {
        // sourceName must be an AZStd::string.
        AZStd::string sourceName(fullSourceFileNameInCallback);
        AssetImporterPlugin::GetInstance()->EditImportSettings(sourceName);
    }});
}```
If you implement an opener in this way, double-clicks and other file open gestures call your opener instead of the operating system.

**Preventing the OS from Opening a Custom File**

You can prevent a source file from being opened by the operating system even if you do not provide a file opener to replace it. For example, you might want to prevent the operating system from opening a file that is in a custom format that the OS cannot handle. To implement this, create an opener for the corresponding asset that has a `nullptr` for the callback.

**Creating a Custom Asset Builder**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Asset Builder SDK lets you develop an asset builder that processes your custom asset type source files into game-ready files. This topic shows you how to create your own asset builder by using the example asset builder that is included with Lumberyard.

To create a builder for a custom asset type:

1. **Create Builder Classes** (p. 319) – Create one or more builder classes that build the asset. The classes must implement the appropriate callbacks and handle shutdown messages from Asset Processor.

2. **Create a Lifecycle Component** (p. 324) – Create a lifecycle `AZ::Component` that registers all of your builder classes. The lifecycle component provides Asset Processor with information to ensure that the correct asset builder is invoked for a file.

3. **Tag Components for Builder Mode** (p. 324) – Tag your lifecycle component and any supporting `AZ::Component` instances with the `AssetBuilder` tag to ensure that they are activated in builder mode.

4. **(Optional) Implement Message Logging** (p. 325) – Use the `BuilderLog()` function and standard `AZ_Trace` macros to log builder-related messages or errors.

This topic describes how to create builder classes, register your builder, tag your components, and implement message logging for your builder.

**Builder Resources**

This topic draws on the following resources, which are included with Lumberyard:

- **Lumberyard Asset Builder SDK** – The Asset Builder SDK enables you to build custom asset processing tools. The source code is located in the following directory:

  `lumberyard_version\dev\Code\Tools\AssetProcessor\AssetBuilderSDK\AssetBuilderSDK`

- **CustomAssetExample gem** – Provides sample custom asset builder code. The builder-related source files for the gem are located in the following directory:

  `lumberyard_version\dev\Gems\CustomAssetExample\Code\Source\CustomAssetExample\Builder`
Prerequisites

This topic assumes that you have a working knowledge of Lumberyard Gems (p. 1073), AZ::Modules (p. 1093), and AZ::Components (p. 974). The next section includes a brief overview of asset builders inside gems.

Asset Builders Inside Gems

Gems contain two kinds of modules:

- A runtime module: `gem_name.dll`
- A tools module: `gem_name.Editor.dll`

These modules contain system components (p. 1098) and tool components. When Lumberyard starts, an AZ::ComponentApplication activates all required system components for the gems that are enabled for the project.

1. Create Builder Classes

To create an asset builder, you must implement one or more builder classes. You can create a builder class with the following steps:

A. Implement the CreateJobsFunction Callback Function (p. 319)

B. (Optional) Declare Source File Dependencies (p. 320)

C. (Optional) Declare Job Dependencies (p. 321)

D. (Optional) Handle Platform-Specific Cases (p. 321)

E. Implement the Callback for ProcessJobFunction (p. 322)

F. (Optional) Declare Product Dependencies (p. 322)

G. (Optional) Declare Product Path Dependencies (p. 323)

H. (Optional) Create a JobCancelListener (p. 323)

I. Shut Down Properly (p. 323)

Each builder class requires two callback functions: one for CreateJobFunction, and one for ProcessJobFunction. Asset Processor uses these callbacks to communicate with your builder class.

Example builder class code is located in the `lumberyard_version\dev\Gems\CustomAssetExample\Code\Source\Builder\CustomAssetExampleBuilderWorker.cpp` file.

A. Implement the CreateJobsFunction Callback Function

In most cases, you should build a JobDescriptor for each processing job for each enabled platform. Then, add the JobDescriptor to the CreateJobsResponse list in the callback for CreateJobsFunction.

Keep in mind the following:

- To ensure that critical files are included, Lumberyard Editor blocks on startup until all callbacks for CreateJobFunction have completed. Due to this startup constraint, we recommend that your code perform minimal work during the CreateJobsFunction callback. For heavy processing work, use the callback for the ProcessJobFunction.
• For extra configurability, you can place arbitrary key–value pairs into the JobDescriptor.m_jobParameters field. The key–value pairs are retained and sent to the callback for ProcessJobsFunction. You can store information gathered during the CreateJobsFunction callback in these key–value pairs and then pass the information as parameters to the callback for ProcessJobsFunction.

• To delete stale products, Asset Processor compares the JobDescriptor that you create with the JobDescriptors that were created in the last iteration. Asset Processor compares JobDescriptors that have the same input source files, PlatformInfo value, and job key.

• You don't need to check whether a JobDescriptor that you create needs to be processed later. Instead, create all possible jobs at every iteration for a particular input asset on each enabled platform.

For an example of a CreateJobsFunction callback, see the CustomAssetExample::ExampleBuilderWorker::CreateJobs() function in the CustomAssetExample gem.

B. (Optional) Declare Source File Dependencies

You can use the Asset Builder SDK to declare dependencies for a source file on other source files. These source files can be any file within the project directory or directories. They do not need to be source files consumed by a builder.

Declaring Asset Dependencies

To declare dependencies, add them in the CreateJobsFunction callback in your builder class to m_sourceFileDependencyList in the CreateJobsResponse structure.

Keep in mind the following:

• Declaring dependencies for a source file implies that the data in its output changes if the files that the source file depends on change. If any of the source dependency files are modified, Asset Processor calls the CreateJobsFunction and ProcessJobsFunction callbacks on the dependent files to reprocess them.

• Asset Processor automatically handles source file dependencies recursively. If the source files downstream emit their own dependencies when they are queried, you do not need to recursively traverse the full tree of dependencies. Emit your local dependencies for each node in the tree, and Asset Processor takes care of the rest.

• Because metafiles such as *.assetinfo files are special case files that cause your asset to rebuild automatically, you do not need to add them as dependencies.

Source File UUIDs versus Paths

The SourceFileDependency structure contains m_sourceFileDependencyPath and m_sourceFileDependencyUUID. Your builder class must supply a value for only one of these fields.

Keep in mind the following:

• If the UUID of the file to add as a dependency is known inside of the CreateJobs method, the builder can simply fill in the m_sourceFileDependencyUUID field. Otherwise, the builder must provide the appropriate source dependency file path for the m_sourceFileDependencyPath field.

• There are two types of path dependencies, indicated by m_sourceDependencyType. These two types are Absolute and Wildcard. Absolute dependencies process only the files that match the provided path exactly. Wildcard dependencies allow for the use of * and % characters in an asset path, which match any number of characters. The use of Wildcard can increase asset build times, so use it only when you must.

• The field m_sourceFileDependencyPath can take both absolute and relative file paths. If a relative path is specified, the appropriate overriding asset is used, if one exists.
• If the builder uses a relative file path for the m_sourceFileDependencyPath field, then the specified path must be relative to one of the watched directories. However, if both the source and the source dependency file exist in the same directory, you can provide the file name without a path.

For an example of adding dependencies inside of a CreateJobsFunction callback, see the CustomAssetExampleBuilderWorker::CreateJobs() function in the CustomAssetExample gem.

C. (Optional) Declare Job Dependencies

You can use the Asset Builder SDK to declare that your custom asset build depends on another job registered with the Asset Processor. Job dependencies are based on either the fingerprint of another job's changes, or the successful completion of a job.

Types of Job Dependencies

There are two types of job dependencies, indicated by the AssetBuilderSDK::JobDependencyType value passed to the constructor for AssetBuilderSDK::JobDependency objects. The dependency type is stored in the m_type member of the created object. These two types are:

• Fingerprint: The Fingerprint dependency causes a job to re-run when the job it's dependent on is reprocessed, and the artifacts generated by the dependent job change according to its fingerprint definition. Fingerprint definitions can include any information, but a fingerprint always includes the state of the source file. This makes fingerprint dependencies a superset of source file dependencies.
• Order: The Order dependency causes a job to be processed whenever the job it's dependent on completes, regardless of whether or not any artifacts or fingerprints have changed. Adding order dependencies reduces the ability to parallelize asset build tasks, so use order dependencies only where needed.

Declare Job Dependencies

Adding job dependencies is done by adding JobDependency objects to an existing JobDescriptor through the m_jobDependencyList member. For example, to add a Fingerprint dependency on the test.example source file for the ExamplePlatform platform, for the job key Example Job:

```cpp
AssetBuilderSDK::JobDescriptor descriptor;
descriptor.m_jobKey = "Example Job"; // Key for matching dependent jobs
descriptor.SetPlatformIdentifier("ExamplePlatform"); // Platform identifier for matching dependent jobs
AssetBuilderSDK::SourceFileDependency sourceFile;
sourceFile.m_sourceFileDependencyPath = "test.example"; // Source file processed by dependent jobs
AssetBuilderSDK::JobDependency jobDependency(
    descriptor.m_jobKey,
    "ExamplePlatform",
    AssetBuilderSDK::JobDependencyType::Fingerprint,
    sourceFile
);
descriptor.m_jobDependencyList.push_back(jobDependency);
response.m_createJobOutputs.push_back(descriptor);
```

D. (Optional) Handle Platform-Specific Cases

CreateJobsRequest provides helper functions for operations related to the enabled platforms. These helper functions can be useful for building the output JobDescriptor for a specific enabled platform.
For more information about declaring, enabling, or disabling platforms, see Configuring the Asset Pipeline (p. 269).

The following functions are available in the Asset Builder SDK. For source code, see
lumberyard_version\dev\Code\Tools\AssetProcessor\AssetBuilderSDK \AssetBuilderSDK\AssetBuilderSDK.*

- HasPlatform(const char* platformIdentifier) - For the specified platform identifier, returns whether that platform is enabled for this CreateJobsRequest. The platform identifier is data driven and user specified. It is usually a string representation of the platform name (for example, "pc" or "osx").
- HasPlatformWithTag(const char* platformTag) - For the specified platform tag, returns whether Lumberyard has any enabled platforms in the CreateJobRequest that contain that tag. Tags are data driven and user specified. They usually identify features that are not specific to a single platform (for example, "mobile" or "console").

E. Implement the Callback for ProcessJobFunction

Asset Processor calls the callback for the ProcessJobFunction when it has a job for the builder class to begin processing. The callback for your ProcessJobFunction should perform the following tasks:

1. Process the source file and perform all work inside the temporary directory.
2. Create at least one product file in the temporary directory.

Keep in mind the following:

- Because the callback occurs on a worker thread, do not spawn threads to do the work in your builder class.
- Do not interact with other threads during the callback. The JobDescriptor that the callback receives is the JobDescriptor that your builder class created in the CreateJobsFunction callback.
- During the callback, create, modify, or write files only in the temporary directory. You can use this temporary directory any way that you want. For example, you can use this directory to write intermediate files and the final products themselves.
- After your job succeeds, Asset Processor copies your registered products to the asset cache. Do not write to the cache directly.

For an example of a ProcessJobFunction callback, see the
CustomAssetExample::ExampleBuilderWorker::ProcessJob() function in the
CustomAssetExample gem in the lumberyard_version\dev\Gems\CustomAssetExample\Code \Source\CustomAssetExample\Builder\CustomAssetExampleBuilderWorker.cpp file.

F. (Optional) Declare Product Dependencies

Product dependencies are used to indicate product files needed at runtime. An example of this kind of dependency is a mesh file that depends on a material. Product dependencies can be referred to by AssetId, path to the source file, or the path to the product. This section describes adding product dependencies based on AssetId references. For information about path dependencies, see the next step, declare product path dependencies (p. 323).

Product dependencies based on AssetId are represented by ProductDependency objects, and may also have an associated 64-bit field flag for metadata. JobProduct objects are assigned dependencies by adding them to the JobProduct.m_dependencies member. If a job's dependencies have their own dependencies, they're correctly handled by the Asset Bundler.
Important
If a product generated by a custom builder is needed at runtime, it must be declared as a product dependency somewhere. The release packaging system relies on product dependencies to figure out what to include in the release build.

G. (Optional) Declare Product Path Dependencies

For situations where you can't use an AssetId to define a product dependency, use either the source or product path to define a product dependency. Where possible, use the AssetId reference system. Product dependencies defined by paths are intended for use with legacy systems and third party tools that don't integrate properly with the AssetId system.

Path dependencies are created as ProductPathDependency objects. These objects are constructed with the ProductPathDependencyType and the string representing the file's path. Path dependencies can be absolute or relative. They are identified either as ProductPathDependencyType::SourceFile for files located in the source folder, or ProductPathDependencyType::ProductFile for product files stored in the cache.

Paths can include the * wildcard character, which matches any number of characters. Wildcards can introduce performance penalties as the Asset Processor evaluates all files of the given ProductPathDependencyType to see if they match the pattern. Additionally, wildcard dependencies are never considered to be resolved and are re-evaluated on each run of the Asset Processor.

JobProduct objects are assigned path dependencies by adding them to the JobDescriptor.m_pathDependencies member.

You can check to see if path dependencies have been resolved by opening the sqlite database in the cache and examining the ProductDependencies table. Unresolved dependencies will have their path in the UnresolvedPath field.

Important
If a product generated by a custom builder is needed at runtime, it must be declared as a product dependency somewhere. The release packaging system relies on product dependencies to figure out what to include in the release build.

H. (Optional) Create a JobCancelListener

Builder classes can use the JobCancelListener function to listen for job cancellation requests in their processJob method. Your code should listen for cancellation requests and then cancel work, if possible, when a request is received. The address of this listener is the job ID of the job that is in processJobRequest. If additional processing like signaling a semaphore or other threading work is required, you can derive from the JobCancelListener and then reimplement Cancel().

For a basic example of JobCancelListener, see the CustomAssetExample::ExampleBuilderWorker::ProcessJob() function in the CustomAssetExample gem in the lumberyard_version\dev\Gems\CustomAssetExample\Code \Source\CustomAssetExample\Builder\CustomAssetExampleBuilderWorker.cpp file.

I. Shut Down Properly

When your asset builder application needs to shut down, Asset Processor uses the address of the registered builder's AZ::Uuid to broadcast the Shutdown() message on the AssetBuilderSDK::AssetBuilderBus. At this point, your builder must stop all tasks and return control to Asset Processor.

Important
Failure to terminate promptly when Asset Processor shuts down and then restarts can cause your system to stop responding. The shutdown message comes from a thread that is separate from the ProcessJob() thread.
2. Create a Lifecycle Component

In this step, create a lifecycle component that registers your builder class with Asset Processor in the Activate() function.

To register each of the builder classes, your lifecycle component must call the AssetBuilderSDK::AssetBuilderBus Event Bus (EBus). This registration allows Asset Processor to send requests to the two registered callback functions in your builder class.

For a code example of a lifecycle component, see the lumberyard_version\dev\Gems\CustomAssetExample\Code\Source\Builder\CustomAssetExampleBuilderComponent.cpp file.

3. Tag Components for Builder Mode

In this step, tag your lifecycle component and all the system components that must be active for your builder worker’s ProcessJobs and CreateJobs functions to work. The components that you tag can be any components that aid in processing your asset type. For example, you might tag a component that contains an implementation of an EBus handler that loads your asset type.

To tag components as mandatory in builder mode, you add an AZ::Edit::Attributes::SystemComponentTags attribute to the AZ::SerializeContext of each component’s Reflect() function. The Asset Builder creates and activates components that you tag as AssetBuilderSDK::ComponentTags::AssetBuilder.

Example

The following example is from the lifecycle component of the CustomAssetExample gem’s CustomAssetExample::ExampleBuilderComponent::Reflect() function.

```cpp
// Perform static reflection or type registration here of any types that you need the serializer to know about
void ExampleBuilderComponent::Reflect(AZ::ReflectContext* context)
{
    if (AZ::SerializeContext* serialize = azrtti_cast<AZ::SerializeContext*>(context))
    {
        serialize->Class<ExampleBuilderComponent, AZ::Component>()
            ->Version(0)
            ->Attribute(AZ::Edit::Attributes::SystemComponentTags,
                        AZStd::vector<AZ::Crc32>({ AssetBuilderSDK::ComponentTags::AssetBuilder }));
    }
}
```

The AZ::Edit::Attributes::SystemComponentTags attribute takes a single AZ::Crc32 value or a vector of AZ::Crc32 values. The example uses a vector. Using a vector makes it more efficient to add tags later in development.

For the source code, see the lumberyard_version\dev\Gems\CustomAssetExample\Code\Source\CustomAssetExample\Builder\CustomAssetExampleBuilderComponent.cpp file.

More About Tagging

Many system components contain logic and startup systems that can be detrimental to asset builds. For example, systems that simulate physics, render GUIs, or attempt to acquire device or network contexts can negatively impact asset builder performance. For this reason, Lumberyard’s Asset Processor loads the same set of gems that Lumberyard Editor loads from the lumberyard_version\dev\project_name\Config\Editor.xml file, but activates only the components that are tagged as mandatory in builder mode. This selective activation of tagged components also makes it possible for an arbitrary number of builder and non-builder components to reside together inside a gem.
4. (Optional) Implement Message Logging

To log builder registration in your lifecycle component, use the following syntax:

```
BuilderLog(AZ::Uuid builderId, const char* message, ...)
```

`BuilderLog` records messages at the application level. To log job-specific messages in the builder classes, use standard `AZ_Trace` macros like the following in the function callbacks in your builder class:

```
AZ_TracePrintf(window, msg)
AZ_Warning(...)
AZ_Error(...)
AZ_Assert(...)
```

This ensures that your messages are in the job’s log file and display in Asset Processor’s log view for that specific job.

Logging Resources

For `BuilderLog` source code, see `lumberyard_version\dev\Code\Tools\AssetProcessor\AssetBuilderSDK\AssetBuilderSDK\AssetBuilderSDK.*`. For trace-related source code, see `lumberyard_version\dev\Code\Framework\AzCore\AzCore\Debug\Trace.*`.

Asset IDs and File Paths

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Consult this section if you are a game engineer who needs to port older game code or develop new code or tools.

File Path Aliases versus Asset IDs

All files accessed for the game runtime go through an interface that supports aliasing of file paths by name. For example, the alias `@ROOT@` refers to the root directory where the `bootstrap.cfg` file is located. If you need to open a file in the root directory, do not go to the root directory or use the current working directory. Instead, use the file name, such as `@root@/filename.cfg`. The various Lumberyard subsystems correctly resolve the alias.

Other aliases available include the following:

- `@log@` – For storing forensic data, such as crashes, logs, traces, performance drops, and unit test output.
- `@cache@` – For storing data that can be cleaned out at any time and does not need to persist.
- `@user@` – For storing data that needs to persist between users. Note that some operating systems may back up this data to the cloud, such as for user preferences.
- `@assets@` – The location of the asset cache. If no alias is specified, this is assumed, so it is almost never necessary to specify this.
- `@devroot@` – The root of your development tree where the editable `engineroot.txt` file is located. This file is shared by many game projects and used by the editor and other tools.
- `@devassets@` – The root of your source asset directory for your game, which is used by the editor and tools.
The following are examples of asset IDs:

- textures/mytexture.dds
- objects/rain/droplet.cfgf
- gamedata.xml
- levels/mainlevel/mainlevel.xml

The following examples are file paths and not assets IDs:

- @assets@/textures/mytexture.dds
- @root@/system.cfg
- C:\dev\mystuff.txt
- \\networkdrive\somefile.dat

The following example is invalid as it mistakenly assumes that the asset cache has the same name as the game and that it is a child folder of the root directory. This isn't true on all operating systems:

- @root@/GameName/textures/mytexture.dds

When referring to assets during runtime, always use the asset ID. Do not prefix asset IDs with @assets@ or the game name, and do not concatenate them with custom strings. Treat asset IDs as immutable data that are not strings and that refer to specific assets. For example, store textures/mytexture.dds, not gems/rain/mytexture.tif.

You can use the FileIO interface, which is accessible through gEnv->pFileIO, to resolve aliased names to full paths if you want to point to an external disk loading tool such as Qt Qfile(). This should almost never be necessary during runtime. If you do use this, your system cannot use remote asset access or support live reloading.

### Converting Asset IDs to Full Paths

If you are writing a new editor tool or porting an existing one from a legacy system, keep in mind the separation between game code and editor code. Game code cannot manipulate asset IDs, and therefore it is invalid to retrieve the game path or concatenate game names with path names. The game code and game modules also have no access to source control, so relying on the game to find out where to save files will not work.

Instead, develop your editor code in such a way that the editor decides where files are saved, and optionally loaded from, and correctly interfaces with source control and the asset processing system. (Source control and asset processing are overhead that is governed by the editor tool, not the game.)

The following utilities and guidelines are provided to make this easier:

- Store only asset IDs for all source assets. For example, if you are writing a file that refers to other files, do not store C:\lumberyard_version\dev\MyGame\myasset.txt in the file's data, for example. Instead, just store myasset.txt, its asset ID.
- If you are in an editor tool, link to EditorCore, and then do the following:
  - #include <PathUtil.h>
  - Call Path::FullPathToGamePath(string) to convert any full path into a game asset ID automatically.
  - Call Path::GamePathToFullPath(string) to convert any asset ID into a full source asset name.
  - Call Path::GetEditingGameDataFolder to see where to save files that do not exist yet, such as for a File Save dialog.
• If you are working in a new system that does rely on legacy systems, you can use an EBus, which has the same functionality as described above. For more information about the EBus, see Event Bus (EBus) in the Amazon Lumberyard User Guide.

• #include <AzToolsFramework/API/EditorAssetSystemAPI.h>

• Call EBus messages ConvertFullPathToRelativeAssetPath and ConvertRelativeAssetPathToFullPath to convert back and forth.

• Call EBus messages GetAbsoluteDevGameFolderPath to get the game directory for File Save dialogs. Use this only when you do not have an asset ID already, such as in the case of new files.

As an example, the following steps code a tool that provides a list of all available assets of type sprite:

To make a list of available sprite assets

1. Use the gEnv->pCryPak file-finding functions to search for all asset IDs. Usually, since @assets@ is assumed, just the directory name or extensions are all that is required, but aliases are accepted.

2. Once you have the asset ID list, call GamePathToFullPath or ConvertRelativeAssetPathToFullPath to convert the list to full source names.

3. Display the appropriate name in the UI, either the real source name or the output name.

4. To edit the file, use the source name to check it out from source control.

5. To save the file, make sure to write it to the source name, not the target name.

6. When the asset compiler recompiles the asset, it notifies you using the asset ID. Make sure you compare the incoming name to this asset ID.

Live Update Messages

If you are on a PC or you are connected to VFS, you can listen for live update messages from the Asset Pipeline and reload your assets when you get them.

To do this, do the following:

• #include <IAssetSystem.h>

• Subscribe a listener to the AssetSystemBus. Subscribers connect through the CRC of the file extensions that they are interested in. Search for "AssetChanged" to see examples in various systems.

Here is an example: BusConnect(AZ_CRC("dds")); // Be notified of all DDS file changes.

Once you get your live reload notification, it contains an asset ID. Consider queueing the request for later if you are in a thread-sensitive module.

Job Dependency for Asset Pipeline

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Manage dependencies for assets so that Asset Pipeline processes them in the correct order.

In Asset Pipeline, you create asset builders using the Asset Builder SDK. The Asset Builder SDK processes your custom asset-type source files into game-ready files. The Asset Builder SDK is a separate executable from Asset Processor. For more information about the Asset Builder SDK, see Creating a Custom Asset Builder (p. 318).
For each asset builder that registered a particular source asset type:

1. The `CreateJobs` operation is invoked, which is where you can declare job dependencies.
2. The `ProcessJob` operation is invoked, which is where your source files are processed into game-ready assets.

A job is uniquely identified by a tuple that contains the following:

- **Job key** – The key that the builder emits for that kind of job in its own `CreateJobs` function.
- **Platform** – The platform identifier of the job. This is generally set to the same platform as the job that runs.
- **Source file** – The source file that the job operates on.

Use the job dependency feature to specify dependencies on jobs, and tell Asset Processor to process jobs for the following conditions:

1. When the fingerprint of another job changes. This is called a *fingerprint job dependency*.
2. When another job completes the processing. This is called *order job dependency*.

**Example 1: Fingerprint Job Dependency**

A fingerprint job dependency means that Asset Processor must process the job if the fingerprint of another job changes. You can specify any number of fingerprint job dependencies.

For **Job A**, specify a fingerprint on **Job B**. So if the fingerprint of **Job B** changes, Asset Processor must reprocess **Job A**.

**Example 2: Order Job Dependency**

An order job dependency means that Asset Processor must process the job only when all other order dependency jobs specified by this job are processed. You can specify any number of order job dependencies.

**Job A** has a specified order job dependency on **Job B**. This means that Asset Processor must process **Job B** first and then **Job A**.

**Example**

You might want a builder that operates on the output of the image compilation set of `example.tif` on the platform `pc`. If so, find the job key of that job (for example, "RC Image") and declare a dependency, like the following: ("RC Image," "pc", "example.tif")

```cpp
enum class JobDependencyType : AZ::u32
{
    //!< Asset Processor should process the dependent job, if there is a change to the fingerprint of the job that it depends on.
    Fingerprint,
    //!< The dependent job should only run after Asset Processor processes the job it depends on.
    Order,
};

//! Job dependency information that the builder will send to Asset Processor.
struct JobDependency
{
    AZ_CLASS_ALLOCATOR(JobDependency, AZ::SystemAllocator, 0);
    AZ_TYPE_INFO(JobDependency, "{93A9D915-8C9E-4588-8D86-578C01EEA388}");
};
```
SourceFileDependency m_sourceFile;
AZStd::string m_jobKey;
AZStd::string m_platformIdentifier;
JobDependencyType m_type;

JobDependency() = default;

JobDependency(const AZStd::string& jobKey, const AZStd::string& platformIdentifier,
const JobDependencyType& type, const SourceFileDependency& sourceFile);

static void Reflect(AZ::ReflectContext* context);

For example code, see the CustomAssetExample gem. For source files related to builders, see the
lumberyard_version\dev\Gems\CustomAssetExample\Code\Source\CustomAssetExample\Builder\CustomAssetExampleBuilderWorker.cpp file.

Example

To declare a job dependency inside a builder, see the following code example.

```cpp
for (const AssetBuilderSDK::PlatformInfo& platformInfo : request.m_enabledPlatforms)
{
    AssetBuilderSDK::JobDescriptor descriptor;
    descriptor.m_jobKey = "Compile Example";
    descriptor.SetPlatformIdentifier(platformInfo.m_identifier.c_str());
    AssetBuilderSDK::SourceFileDependency sourceFile;
    sourceFile.m_sourceFileDependencyPath = "test.examplesource";
    // We are declaring a fingerprint job dependency for this job on any job of
    "test.examplesource" which has job key "Compile Example " for every enabled platforms
    AssetBuilderSDK::JobDependency jobDependency("Compile Example",
    platformInfo.m_identifier.c_str(), AssetBuilderSDK::JobDependencyType::Fingerprint,
    sourceFile);
    descriptor.m_jobDependencyList.push_back(jobDependency);
    response.m_createJobOutputs.push_back(descriptor);
}
```

Note

If Asset Processor detects a cyclic job dependency, it displays a warning message and then
processes the first job to unblock the remaining jobs.

Raw File Access in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

This topic describes how to directly access files in Lumberyard for special use cases. However, it's
recommended that you use the Lumberyard Asset system to work with asset files. In most cases,
raw file access is not required. For more information see Working with the Asset Pipeline and asset
files (p. 247).

When you write an AssetHandler-derived class to load assets in Lumberyard, runtime file handling
is automatic. However, some cases might require lower levels of file access at run time. Scenarios that
might require low-level file access include:

- Loading raw configuration files from the deployment root during startup before .pak files are
  mounted and available.
• Direct access to the files in a .pak file.
• Direct access to files at arbitrary locations on disk.
• Streaming file formats (for example, audio or video) that do not load the entire file but play it back. This approach commonly uses middleware to play back or capture audio, video, or vertex data. Most such systems require that you implement file hooks to perform operations like read, seek, tell, open, and close. In these cases, direct file access might be easier than treating the files as assets and performing operations on them with AZ::Data systems.
• Other legacy systems or middleware that require direct file access for which AZ::DataAsset systems cannot be used. However, note that it is possible to write a file access shim that describes how to access files for most middleware.
• Loading raw source files from locations other than the asset cache. Loading source files from locations other than the asset cache is possible only for tools inside Lumberyard Editor. Only the asset cache ships with your game, so loading raw source files from locations other than the asset cache at run time is not possible.

The few cases where you need to work directly with files are covered by a small number of classes in the AZ::IO namespace such as FileIOBase and FileIOStream.

The FileIOBase Virtual Class

The pure-virtual base class FileIOBase (located in \dev\Code\Framework\AzCore\AzCore\IO AzCore\IO\FileIO.h) defines the API for accessing files. It is a basic blocking low-level file API, as the following code shows:

```cpp
class FileIOBase
{
    ...
    static FileIOBase* GetInstance();  ///< Use this to get a concrete instance of the API that follows.
    ...
    virtual Result Open(const char* filePath, OpenMode mode, HandleType& fileHandle) = 0;
    virtual Result Close(HandleType fileHandle) = 0;
    virtual Result Tell(HandleType fileHandle, AZ::u64& offset) = 0;
    virtual Result Seek(HandleType fileHandle, AZ::s64 offset, SeekType type) = 0;
    virtual Result Read(HandleType fileHandle, void* buffer, AZ::u64 size, bool failOnFewerThanSizeBytesRead = false, AZ::u64* bytesRead = nullptr) = 0;
    virtual Result Write(HandleType fileHandle, const void* buffer, AZ::u64 size, AZ::u64* bytesWritten = nullptr) = 0;
    virtual Result Flush(HandleType fileHandle) = 0;
    virtual bool Eof(HandleType fileHandle) = 0;
    ...
};
```

The FileIOBase class contains operations to find files, create or delete directories, and inspect attributes. In addition, the class contains a directory aliasing system that is covered later in this document.

Getting an Instance of the I/O Interface

For almost all file operations, you call the GetInstance function to retrieve an instance of the file I/O interface:

```cpp
AZ::IO::FileIOBase::GetInstance()
```

Notes

• All file operations in the FileIOBase class are blocking.
• FileIOBase file operations behave similarly to the C language API fopen, fread, and fclose operations, but are 64-bit aware and work with very large files.
• Because the FileIOBase instance is created and initialized when the engine initializes, it is generally always available. It can inspect .pak files and arbitrary files on disk. For more information, see The FileIO Stack (p. 333) later in this document.

Note
Because .pak files are initialized only after the application boots, attempting to access data inside .pak files before they are mounted will fail.

For more information, see the code comments in the FileIO.h file.

The Aliasing System

In addition to a set of file functions mentioned above, the FileIOBase class provides directory aliases. Directory aliases are prefixes that you add to a file name. An alias indicates a virtual directory for a .pak file or arbitrary location on disk.

Note
We recommend that you always use the aliasing system to refer to files that are in the cache. Never use absolute paths. Files in the cache might be inside .pak files or in unexpected locations on mobile devices. In these cases, the use of absolute path names can fail.

Getting the Path of an Alias

To retrieve the path associated with an alias, use the GetAlias function.

FileIOBase::GetAlias()

List of Aliases

This section describes the use of directory aliases.

@assets@

Refers to the asset cache. This is the default alias when no alias is specified. Note the following:
• Because @assets@ is the default alias, code can simply load files by name (for example, textures\MyTexture.dds) without using the asset system. This makes it unnecessary to have the @assets@ alias appear throughout the code.

Note
If you are loading files from the asset cache, do not prefix your file names with the @assets@ alias. The use of aliases is required only when you must alter the default behavior. This best practice makes your code easier to read and enhances compatibility.
• During development on a PC, @assets@ points to the dev\Cache \<game_name>\pc\<game_name> directory. After release, it points to the directory where your .pak files are stored (not the root of your cache where your configuration files are stored).
• Because the asset cache can be locked by asset processing operations, attempting to write to the asset cache can cause an assertion fail. Do not attempt to write files to the asset cache.

@root@

Specifies the location of the root configuration files like bootstrap.cfg. Note the following:
• The asset cache @assets@ can be a child directory of @root@, but that is not always the case. Therefore, do not make this assumption. If you want to load a root file, use @root@. If you want to load assets, either use no alias (because @assets@ is the default), or use @assets@.
During development, the @root@ directory maps to your dev\Cache\<game_name>\pc directory. In release, this directory is the root directory of your game distribution (where the bootstrap.cfg file is stored).

Attempting to write to the @assets@ location causes an assertion fail. You should change these files in your source dev\ directory, not in the cache.

@user@

Specifies a writable directory that stores data between gaming sessions. Note the following:

- It is not expected that the user will delete this directory.
- On a PC, @user@ is the dev\Cache\<game_name>\pc\user directory.
- On other operating systems and devices, the @user@ location can be different. Some mobile operating systems have restrictions on where applications can write files.

@cache@

Specifies a writable directory for storing temporary data. The user can delete this data at any time.

@log@

Specifies a writable directory for storing diagnostic logs.

**Code Examples**

A. The following code example opens a file in the assets directory.

```cpp
using namespace AZ::IO;
HandleType fileHandle = InvalidHandle;
// Because @assets@\config\myfile.xml is desired, an alias doesn't have to be specified. // All files in the @assets@ alias are always lowercase. This removes concerns about // case sensitive environments.
if (FileIOBase::GetInstance()->Open("config/myfile.xml", AOpenMode::ModeRead| AOpenMode::ModeBinary, fileHandle))
{
    // Open succeeded. Use other API operations of FileIOBase to perform operations with
    // the handle. Remember to close the file!
    FileIOBase::GetInstance()->Close(fileHandle);
}
```

Note that because aliases are used in the preceding example, the config\myfile.xml file would be found even if it is inside a .pak file.

B. The following code example opens a file in the log directory and appends log lines to it.

```cpp
using namespace AZ::IO;
HandleType fileHandle = InvalidHandle;
// In this rare case, you want to write to a file in the @log@ alias, // so the file name must be specified.
// Because you're writing a file to a non-@assets@ directory, it can contain case.
if (FileIOBase::GetInstance()->Open("@log@/gamelog.txt", AOpenMode::ModeAppend|AOpenMode::ModeText, fileHandle))
{
    // Open succeeded. Use other API operations of FileIOBase to perform operations with
    // the handle.
    FileIOBase::GetInstance()->Close(fileHandle);
}
```

The FileIOStream class in the AZ::IO namespace automatically closes a file when it goes out of scope and presents it as a GenericStream interface. This provides compatibility for systems such as the streamer system and serialization system that expect generic streams.
Tools-Only Aliases

The following aliases are applicable only for editor tools.

@devroot@

Specifies the \dev\ directory of your source tree where files like bootstrap.cfg are located. These files are consumed by the Asset Processor and deployed into the cache specified by @root@.

@devassets@

Specifies the location of your game project's assets directory in the source tree. This directory contains uncompiled source files like .tif or .fbx files. It does not contain compressed .dds files or other assets that a game normally uses. Note the following:

• @devassets@ is a good starting point for a file open dialog that asks a user where to save a new file.
• Because existing files might be in a gem, do not save them in @devassets@. Instead, when your editor opens a file, have your editor remember the file's location. Then have the editor save the file to the file's original location.
• Because not all source files are located in @devassets@ (many are located in gems), do not attempt to find all source files by searching its location.

The FileIO Stack

To service the needs of the game client and tools, more than one FileIO instance is created. These instances form a stack through which file requests flow, as the following diagram illustrates.

The behavior of the Either/Or branch depends on whether the virtual file system (VFS) feature (RemoteFileIO in the diagram) is enabled. VFS reads assets remotely from non-PC devices such as Android (p. 3174) and iOS (p. 3208). VFS is required for live reloading of assets (p. 291). Otherwise, assets would need to be deployed directly onto game devices. VFS is disabled by default. To enable VFS, edit the remote_filesystem entry of the \dev\bootstrap.cfg configuration file, as in the following example.
Because the VFS feature is at a low level, file access operations are transmitted transparently over the network through the Asset Processor to the layers above.

To send requests for files through other systems, you can implement your own version of FileIOBase (or one of the derived classes such as RemoteFileIO or LocalFileIO). If you replace the instance returned by either GetInstance or GetDirectInstance with your own instance, the FileIO system uses your layer instead. You can form a stack of additional filters by replacing the instance with your own. Then make your own instance call down into the previously installed instance.

**Asynchronous Streaming**

If you want to use asynchronous background streaming, consider using the AZ::IO::Streamer class instead of FileIOBase. The Streamer class uses FileIOBase internally, but it uses asynchronous semantics. To use the Streamer class, you pass data and a deadline to it. The Streamer class puts the data into a buffer and does its best to fulfill the request before the specified deadline.

For more information, see the code and comments in the \dev\Code\Framework\AzCore\AzCore\IO\Streamer.h file.

---

### Bundle game assets in Amazon Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Amazon Lumberyard provides a set of tools for intelligently bundling assets and managing your game's product and source dependencies. Use these tools to package only the assets currently used in your game project and reduce the deployment size and complexity of your product overall.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What is the Amazon Lumberyard Asset Bundler?</strong> (p. 335)</td>
<td>Learn how the Lumberyard Asset Bundler tools work and how you can use them in your game project.</td>
</tr>
<tr>
<td>Bundle assets for release (p. 344)</td>
<td>Learn how to create release builds and use the Asset Bundler to package assets.</td>
</tr>
<tr>
<td>Creating Multiple Asset Bundles (p. 349)</td>
<td>Learn how to create multiple asset bundles for a game project.</td>
</tr>
<tr>
<td>Creating Content Patch Bundles (p. 354)</td>
<td>Learn how to create delta content patches for your shipped games.</td>
</tr>
<tr>
<td>Migrating to the New Asset Bundler (p. 359)</td>
<td>Learn how to migrate projects from versions prior to Beta v1.22 to the new asset bundler system.</td>
</tr>
<tr>
<td>Resolving Missing Assets (p. 363)</td>
<td>Learn how to resolve missing assets and their dependencies.</td>
</tr>
<tr>
<td>Verifying Asset Bundles (p. 365)</td>
<td>Learn about the tools and processes for managing and verifying your game project's assets.</td>
</tr>
</tbody>
</table>
What is the Amazon Lumberyard Asset Bundler?

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Asset Bundler is a command-line tool, AssetBundlerBatch.exe, and a set of specific file formats generated and used by the tool.

Prerequisites to use the Asset Bundler

To use the Asset Bundler, your game project must meet the following criteria:

- You have Lumberyard version 1.22 or later installed.
- You have game project in Lumberyard version 1.22 or later. This is the project that you want to generate asset bundles for.
- The assets that you are bundling have been processed by the Asset Processor (p. 248).
- You have Visual Studio 2017 or Visual Studio 2019 (any edition) installed and configured for C++ development.

Why use the Asset Bundler?

With the Asset Bundler, you don't have to figure out which assets that you need to include in your game and which assets to exclude. This can help save you time. For example, you have a working folder with 10,000 assets, but your final game uses only 5,000 of those assets. You want to ship only those 5,000 assets. Tracking this manually can quickly become complicated and expensive.

The Lumberyard Asset Bundler helps make shipping the specific assets used for the release of your game more reliable and repeatable. The reliability is based on an underlying dependency system that informs Asset Bundler which assets to include. If you make changes to your game and add or remove assets, the next time that you run the Asset Bundler, it automatically includes or excludes these new assets for you. The repeatability is based on underlying configuration files that provides consistency across different runs of the Asset Bundler.

By using the Asset Bundler, because you have smaller release packages, less risk of shipping unused or inappropriate assets, and more effective asset management.
How do I use the Asset Bundler?

Generating a platform-specific (for an iOS-specific release, or a console-specific one) asset bundle using the Asset Bundler follows these general steps:

1. **Define your seeds.** A *seed* is generally a top-level asset, such as a .pak file that contains an entire game or game level. You must have one or more seeds defined in a seed list. A seed list is a file with the suffix .seed.

2. **Generate your asset lists.** The Asset Bundler evaluates each seed defined in your seed list. It then recursively evaluates each dependent asset for inclusion in a complete asset list. Generated asset lists have the suffix .assetlist.

3. **Configure your bundle settings.** Provide a configuration file, called the "bundle settings file," which has the suffix .bundlesettings. It contains all of the specific settings, such as the maximum bundle size, when creating a release bundle for your game.

4. **Bundle your assets.** The Asset Bundler utility, AssetBundlerBatch.exe, uses the asset list and bundle settings files to create a release bundle as a .pak file.

The following diagram illustrates the relationships and process for bundling assets with the Asset Bundler.

In this example, the seeds are the assets Level1.pak and Level2.pak. These .pak files reference Lumberyard slice files, which reference the entity meshes they contain, which in turn reference the material and texture files for those entities.

With those product dependency relationships in place, the Asset Bundler examines the hierarchies of each seed and generates an asset list. Asset lists, along with the bundle settings file that you create, are used to assemble the final bundle as a .pak file with all of the dependent assets. Any assets not associated with a seed are not included in the final release bundle.

The Asset Bundler runs whenever the Asset Processor starts, which includes any time you launch Lumberyard. You can also run it from the command line with the AssetBundlerBatch command. For more information on the latter, read the Asset Bundler Command Line Reference (p. 394).

To get started using the Asset Bundler, read the following tutorials:
• **Work with Asset Bundling and the Lumberyard Starter Game (p. 344).** This tutorial uses the Starter Game packaged with Lumberyard and walks you through the general asset bundling process.

• **Creating Basic Asset Bundles (p. 344).** This tutorial covers the most simple case for asset bundling and release, using a single seed. Once you've worked through this topic, read the extended tutorials for multiple packages (p. 349) and delta patches (p. 354).

### Why define product dependencies?

Many game projects restrict the content in their bundled builds. This is often done to hit target build sizes, such as the over-the-air download cap on iOS. It can also restrict content from builds that you really don't want in your release, such as content that would result in a negative ESRB rating for your game. The product dependency system provides tools to help you understand why assets are included in your asset bundles. This lets you debug the associated references and remove them if you don't want an asset.

There are two types of asset dependencies:

- **Product dependencies.** This type of dependency is between two product assets. It declares that there is a specific link between these two assets, and in most cases you'll want both available in your final packaged build of your game.

- **Source dependencies.** This type of dependency indicates that a source asset must be processed first, and that information from processing the source asset is required before the final product asset can be generated. For example, a process in an asset pipeline that runs on `Tree.fbx` has declared a source dependency on `Tree.mtl`. The `Tree.fbx` job won't run until the `Tree.mtl` job has completed, which guarantees that product details for `Tree.mtl` are available when the `Tree.fbx` job runs, and that information can be used in the generation of `Tree.fbx`'s product (in this example, a single CGF file, `Tree.cgf`).

In these topics, we are referring to **product dependencies** unless otherwise indicated.

The Asset Bundler also lets you exclude content that you don't want to release using the asset list comparison feature (p. 380).

### How product asset dependencies are used by Lumberyard

Product dependencies generate asset lists from seed lists. This simplifies the management of what goes into your packaged game content for your game releases. These dependencies are expressed as paths (either relative or absolute, and can optionally include the wildcard *) and are managed by an Asset Builder during processing. Within the Asset Builder, the `m_pathDependencies` variable contains the list of dependency paths.

For example, you have a game that has one level for your release build. Your game contains a single tree. You also have a separate test level that you don't need to ship, which contains a single rock.
Overview

In the asset bundling workflow in versions of Lumberyard prior to version 1.22, each file had to be tracked and added to the game package. This was generally done by specifying wildcards. It worked for simple builds. However, this process became hard to use when a game was split into multiple bundles. The multiple bundles accommodated content being downloaded on demand, or the hard upper limit on package size because of platform package restrictions. It also became difficult to safely manage excluded content, such as when your game project had assets that could not appear in your release build.

Based on the file layout in the previous example, here's a simple diagram of the workflow in Lumberyard versions prior to 1.22:

For more complex game projects, the asset bundling workflow can look more like this:
The bundling workflow in Lumberyard version 1.22 has fewer steps, fewer loopbacks, and less complex steps than prior versions.

The following example shows a basic workflow for a simple game with assets that emits dependencies correctly.

For more complex game projects, with file loads implemented in C++ and custom asset builders, the asset bundling workflow can look more like this:
New Bundling System
Package release assets only: full process

In the previous bundling method, it was necessary to learn how each asset worked to find asset references in Lumberyard's tools. With the new asset bundling workflow, you need only dig into the individual asset systems when a problem occurs using the missing asset scanning tools (p. 365). You also have opportunities earlier in development to verify that you are bundling the correct assets. This is unlike the previous method that offered verification tools only for a bundled release build.

Asset Builders and the Asset Bundler

An Asset Builder is a standalone application that primarily translates intermediate assets to a platform-native asset format. An Asset Builder also provides dependency tracking and tasks related to managing asset metadata.

Lumberyard ships with Asset Builders for many common asset types. The Asset Bundler relies on the information produced by the Asset Builders for your project to manage the dependencies. If you use custom asset types, you can create your own Asset Builders (p. 318) to support proper asset management and bundling with Lumberyard.

For example, Asset Builders for images can convert any portable image asset into a set of performance image formats. When the Asset Builder for the images run, they create the dependency tree for the performance-oriented image formats generated from the more general-use portable image formats, and define them as dependent on the respective portable image format. If you create the performance-oriented images manually or through a separate process, the dependencies are not defined and not available to the Asset Bundler.
In addition to processing your asset, Asset Builders also determine any product or source asset dependencies and store that information in an Asset Catalog (p. 308) for later use by the Asset Bundler. Specifically, “defining asset dependencies” means updating your custom Asset Builder to identify all of the other assets that the asset being processed depends on. It is important to define asset dependencies so that you can perform accurate Asset Bundling to ship your game. Without defined dependencies the Asset Bundler has no way to know which assets your game needs when it's time to prepare your asset bundles release—you could end up missing assets, or include too many, or ship undesirable ones. With asset dependencies defined in your Asset Builder(s), you can be assured that you are including exactly the assets you need for your game and nothing more.

You can use the Asset Builder APIs to develop your own Asset Builders, which can then process your custom asset type's source files and generate files that you can use in Lumberyard. Source code for a sample Asset Builder implementation is provided in the Games/CustomAssetExample/Code/Source/Builder directory under your Lumberyard installation root.

For more information, see the Asset Builder documentation (p. 318).

### Define dependencies in your Asset Builder

In addition to processing your asset, Asset Builders also define any product or source asset dependencies, and then store that information in a database for later use by the Asset Bundler. Specifically, defining an asset dependency means updating your custom Asset Builder to identify all of the other assets that the asset being processed depends on. It is important to define asset dependencies so that you can perform accurate Asset Bundling to ship your game. Without defined dependencies, the Asset Bundler cannot identify which assets your game needs when it's time to prepare your asset bundles release. As a result, you could end up missing assets, including too many, or shipping undesirable ones. With Asset Dependencies defined in your Asset Builders, you are including the exact assets that you need for your game.

### Asset bundling and source control

During the asset bundling process, you can generate multiple different types of artifacts. We recommend that you track some of these in source control, as per the suggested best practices in the following table.

#### Asset bundling artifact storage best practices

<table>
<thead>
<tr>
<th>File Extension</th>
<th>Source Control Details</th>
<th>File Storage Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>.seed</td>
<td>Keep your seed lists in source control.</td>
<td>If you have multiple builds and are patching bugs in older revisions of your game, consider keeping versioned seed lists that match each release. For example, you could keep seed lists for multiple releases on Steam, such as an unstable feature release and your stable release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If your game has one release for a specific platform and your release pipeline is relatively uncomplicated, you can maintain one version of a seed list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You can also track your game's seeds by splitting the contents</td>
</tr>
<tr>
<td>File Extension</td>
<td>Source Control Details</td>
<td>File Storage Best Practices</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>.assetlist</td>
<td>Keep your asset lists in source control. This makes it more efficient to generate delta patches for your game. These are patches that contain only modified files since your last release.</td>
<td>You should store a separate asset list file in your source control for each release that you generate from them. Asset lists contain fingerprint information for each file. If you want to generate a delta patch and you don't have your asset list from the previous version of your game, you must sync your source control back to the exact time you generated that first asset list. Storing each asset list for a version separately can save you time later. For more information, see Asset Bundling &amp; Release Build Tutorial - Delta Bundles, Patching Content.</td>
</tr>
<tr>
<td>.comparisonrules</td>
<td>If you use comparison rules, keep them in source control.</td>
<td>Storing your comparison rules in a file is an optional step in the asset bundling workflow. Use comparison rule files to help make sure that you're generating your packaged game content in the same way across multiple releases.</td>
</tr>
<tr>
<td>.bundlesettings</td>
<td>If you use bundle settings, keep them in source control.</td>
<td>Storing your bundle settings in a file is an optional step in the asset bundling workflow. Use bundle settings files to help make sure that you're generating your packaged game content in the same way across multiple releases.</td>
</tr>
</tbody>
</table>
In the following example, there is a game that has two sets of bundled content: Assets loaded by the engine itself, and assets loaded by the game. In this scenario, the game has just release a version 2 update. Bundles have been created that upgrade existing version 1 players to version 2; and version 2 only bundles content for new players who don't have version 1 content.

The following table provides some examples of the artifacts generated and used in bundling for this example scenario and how they are used.

### Example file artifacts in an Asset Bundler workflow

<table>
<thead>
<tr>
<th>Files (Example)</th>
<th>Usage</th>
<th>Notes</th>
</tr>
</thead>
</table>
| • EngineContent_v1.assetlist  
• Game_engineRemoved_v1.assetlist | The asset lists used to generate the bundles for version 1 of the game. | You should keep these files in source control with the version in the file name. This lets you use these files to generate delta patches for future game releases. |
| • EngineContent_v1_to_v2.assetlist  
• EngineContent_v2.assetlist  
• Game_engineRemoved_v2.assetlist  
• Game_engineRemoved_v1_to_v2.assetlist | Asset lists used to generate bundles for version 2 of the game. | You should keep these files in source control with the version in the file name. This allows these files to be easily used to generate delta patches for future game releases. |
| • Game_all_v1.assetlist  
• Game_all_v2.assetlist | These are asset lists generated as an interim step in creating the game bundle. An asset list comparison that removes content in the engine asset list is used to trim these down to the "engineRemoved" asset lists in the previous row. | You may have asset lists that you generate as interim steps that aren't used to create bundles directly. These could go into source control as a historical record, but they won't be used in the future to generate delta patches. |
| • DeltaBetweenVersions.comparisonrules  
• Game_RemoveOverlappingEngineContent.comparisonrules | Comparison rules files should be stored in source control, so you can use the same bundle creation process each release. Versioning information is not stored in the file name, because | |

Don't store .pak files for your asset bundles in standard source control. Keep them in a separate secured file store.

Keeping track of your previously released content can help you debug problems. Asset bundles can be large, so diffing them isn't particularly useful. We recommend that you find an alternate place to store these files. If you're using Git as your source control solution, Git LFS can provide a useful home for these files. You can also use Amazon S3 to store your historical asset bundles.

**File Extension**
- .pak (asset bundles)

**Source Control Details**
- Don't store .pak files for your asset bundles in standard source control. Keep them in a separate secured file store.

**File Storage Best Practices**
- Keeping track of your previously released content can help you debug problems. Asset bundles can be large, so diffing them isn't particularly useful. We recommend that you find an alternate place to store these files. If you're using Git as your source control solution, Git LFS can provide a useful home for these files. You can also use Amazon S3 to store your historical asset bundles.
### Bundle assets for release

<table>
<thead>
<tr>
<th>Files (Example)</th>
<th>Usage</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GameBundleSettings.bundlesettings</td>
<td>The settings used to generate the bundles.</td>
<td>Bundle settings files should be stored in source control, so you can use the same bundle creation process each release.</td>
</tr>
<tr>
<td>Game.seed</td>
<td>The game's seed list.</td>
<td>Your game's seed list should be stored in source control. As you change your game's content, you'll add and remove seeds from it. Versioning information is not stored in the file name, because it's unlikely you will need to use two versions at the same time. If you do, rely on your source control's history to retrieve older versions.</td>
</tr>
<tr>
<td>Engine_v1.pak</td>
<td>The asset bundles generated for each release.</td>
<td>Keep track of your game content's packaged asset bundles, to use in your game's releases. You can store your game's packaged content in source control, but this is not optimal, because these are large, binary files. We recommend that you use a separate storage solution, like Amazon S3.</td>
</tr>
<tr>
<td>Engine_v1_to_v2.pak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Engine_v2.pak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game_v1.pak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game_v1_to_v2.pak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game_v2.pak</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Additional Resources

- Asset Bundler Concepts and Terms (p. 391)
- Work with Asset Bundling and the Lumberyard Starter Game (p. 344)
- Creating Basic Asset Bundles (p. 344)
- Creating Multiple Asset Bundles (p. 349)
- Creating Content Patch Bundles (p. 354)
- Migrating to the New Asset Bundler (p. 359)

### Build and bundle assets for release in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This tutorial guides you through the process of building the code and assets to release a Lumberyard project, using the Starter Game sample project (p. 148). You'll learn how to:
• Create a release build of your game's executable.
• Set up the directory structure of a release build.
• Compile shaders and generate shader paks.
• Compile auxiliary data, like configuration information and gem assets.
• Create bundled content using the asset bundling system.
• Run a stand-alone release build for your project.

Prerequisites

To complete the procedures in this tutorial, you need the following:

• Amazon Lumberyard. Download the latest version of Amazon Lumberyard.
• (Recommended) Some familiarity with the Asset Bundler concepts and terminology (p. 391). This tutorial uses seed lists and asset lists to generate bundles.

Configure Lumberyard to build the Starter Game project

1. Open the Amazon Lumberyard Project Configurator.
2. Select Starter Game and then select Set as default in the upper-right corner of the Project Configurator screen.
3. Open a command prompt and navigate to the Lumberyard root directory at `lumberyard_dir\dev`.
4. Configure the build system for Starter Game and generate configuration files:

```
lmbr_waf configure
```

### Create a release build

1. Open a command prompt and navigate to the Lumberyard install root directory.
2. Create a profile build using the `all` build spec. Depending on your hardware, this can take a while.

```
lmbr_waf build_win_x64_vs2019_profile -p all
```

   This step ensures that your editor, asset processor, asset builders, and other edit-time content is up to date. It’s also required for shader generation.

3. Make a release build with the `game_and_engine` build spec. Depending on your hardware, this can take a while, but should be faster than a full profile build.

```
lmbr_waf build_win_x64_vs2019_release -p game_and_engine
```
Create a directory structure for the game release

1. Open a command prompt and navigate to the Lumberyard install root directory.
2. Create a directory for your game release. The following command creates the release directory:
   ```
   mkdir %USERPROFILE%\StarterGameRelease
   ```
   **Note**
   You can create this directory anywhere you want, but the rest of this tutorial assumes the release directory is in this location. It’s not recommended that you create this directory anywhere inside of your Lumberyard install. If you do, detecting missing assets and diagnosing bundle problems becomes more difficult.
3. Create a subdirectory that will contain the game binaries and libraries:
   ```
   mkdir %USERPROFILE%\StarterGameRelease\release
   ```
4. Copy the contents of the release build into the StarterGameRelease\release directory:
   ```
   xcopy /s Bin64 vc142.Release %USERPROFILE%\StarterGameRelease\release
   ```
   If prompted **Does path specify a file name or directory name on the target?**, choose **Directory**.
   **Note**
   Release builds include some metadata like debug symbols in a .pdb file. When releasing your game, make sure to delete any compiler metadata that's copied over that isn't needed for launching or running your game.
5. Create a subdirectory that will contain the game data:
   ```
   mkdir %USERPROFILE%\StarterGameRelease\startergame
   ```
   The remaining steps in this tutorial show how to build and copy your game data to this directory.

Export level data

1. Open the Lumberyard Editor and load the Starter Game level by selecting **File > Open Level** (Ctrl+O) and selecting the **SinglePlayer** level.
2. Select **Game > Play Game** (Ctrl+G) from the Editor’s main menu to enter Game mode. Roam through the level to load the shader assets. Make sure that you view as much of the level as possible, in order to load them all.
   
   You can also load shaders by flying the camera through the editor’s viewport, but make sure that you load shaders around the player’s starting area. Otherwise, running the standalone game executable will show a black screen.
3. Select **Game > Export to Engine** (Ctrl+E) from the Editor’s main menu to export the level data to a .pak.

Generate shaders and auxiliary data

1. Open a command prompt and navigate to `lumberyard_dir\dev\Tools\CrySCompileServer\x64\profile`.
2. Start the shader compiler. **Don't close the command prompt.**
3. Open a second command prompt and navigate to the Lumberyard install root directory.

4. Compile and package the shaders:

   `lmbr_pak_shaders.bat StarterGame D3D11 pc`

   After building the shaders, close the command prompt window where `CrySCompileServer.exe` is running.

5. Copy the shader .pak files generated by the compiler into the game data folder:

   `copy build\pc\StarterGame\* %USERPROFILE%\StarterGameRelease\startergame`

6. Generate the game's auxiliary data:

   `Tools\Python\python3 BuildReleaseAuxiliaryContent.py --platforms pc --buildFolder Bin64vc142`

   The auxiliary data includes configuration information for the engine and game loading and level data.

7. Copy the auxiliary data to the release directory:

   `xcopy /s startergame_pc_paks %USERPROFILE%\StarterGameRelease`

---

**Generate game asset bundles**

1. Open a command prompt and navigate to the Lumberyard install root directory.

2. Bundle assets needed by the game engine:

   `Bin64vc142\AssetBundlerBatch.exe assetLists --addDefaultSeedListFiles --assetListFile engine.assetlist`

   `Bin64vc142\AssetBundlerBatch.exe bundles --assetListFile engine_pc.assetlist --outputBundlePath %USERPROFILE%\StarterGameRelease\startergame\engine.pak`

   This generates the `engine_pc.pak` file in your release folder. The engine pak contains the assets required by the engine and gems.

3. Bundle game content and level assets:

   `Bin64vc142\AssetBundlerBatch.exe assetLists ^ --addSeed Levels\Game\SinglePlayer\level.pak ^ --addSeed project.json ^ --addSeed gems.json ^ --addSeed scripts/ai/navigation.xml ^ --assetListFile startergame.assetlist`

   `Bin64vc142\AssetBundlerBatch.exe bundles --assetListFile startergame_pc.assetlist --outputBundlePath %USERPROFILE%\StarterGameRelease\startergame\ startergame.pak`

   This generates the `startergame_pc.pak` file in your release folder.

   **Important**

   `--addSeed` takes a path relative to your project folder (for source assets) or the asset cache (for product assets). For Starter Game, project source assets are located in
lumberyard_dir\dev\StarterGame. Don't use absolute paths or paths relative to the current directory when adding a seed.

Run your packaged release

1. Open a command prompt and navigate to your packaged release at %USERPROFILE%\StarterGameRelease.
2. Run the launcher executable for your game and load the map:

```
release\StarterGameLauncher.exe +map singleplayer
```

If your content bundles are correct, the starter game will load and be playable. Use Alt+F4 to exit the game.

If objects are displayed but textures are missing, it probably means you forgot to export the level before packaging assets or didn't add the level.pak file as a seed. You could also be missing some shaders - run through the level and build the shader package again.

If the test isn't successful, common issues may occur. For example, error messages may display, the launcher may shut down, or a black screen displays. For more information about troubleshooting common issues, see Resolving Missing Assets (p. 363) and Compiling Shaders for Release Builds (p. 291).

**Note**

When you run the release build, it creates a User subdirectory under your release build directory. Be sure to delete this directory before shipping the release build.

Next Steps

Now that you've learned the basics of bundling assets for release, go on to further reading:

- Learn about how bundles are mounted, so that you can load content dynamically. See Creating Multiple Asset Bundles (p. 349).
- Explore the asset bundler functionality. See Lumberyard Asset Bundler Command-Line Tool Reference (p. 394).
- Learn how to scan for missing dependencies in your bundles. See Using the Missing Dependency Scanner (p. 366).
- To ask questions about the Asset Bundler and get support, see the Amazon Lumberyard forums.

Creating Multiple Asset Bundles

This tutorial is out of date for the new Starter Game project that shipped as part of Lumberyard 1.25. We're working to revise it and provide better, more comprehensive instructions.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Build and bundle assets for release in Lumberyard (p. 344) tutorial gets you started with using the asset bundling system to produce a game release. However, it doesn't represent what most games
require. In this tutorial, you learn the bundling process for a common use case: Games that download additional content after the player launches the game. The tutorial shows you how to create a release build with multiple game levels in separate asset bundles that have the contents of the base game removed.

This topic covers the following points:

• Adding a new level and basic content to the Starter Game project.
• Creating multiple asset bundles separated by game level.
• Running the new build with and without the additional bundles.

Prerequisites

This tutorial requires completing Build and bundle assets for release in Lumberyard (p. 344). After finishing that tutorial, you should have:

• A bundled release of Starter Game located at %USERPROFILE%\StarterGameRelease
• The startergame_pc.assetlist and engine_pc.assetlist asset list files used to generate the starter game asset bundles.

It's also useful to be familiar with the Amazon Lumberyard Asset List Comparison Operations (p. 380) for this tutorial.

Create a second level

In this section of the tutorial, you'll create a new level for the Starter Game that displays a static scene containing a single entity.

1. Launch the editor, and create a second level. Name it level2.

2. Create a camera. Right-click on the viewport and choose Create camera entity from view.
3. Create an object in front of the camera. To make sure that the object you create is located correctly, you may need to pull the perspective in the viewport back a bit so that you can clearly see the view frustum of the camera and know where the place the object.

   a. Create a new entity in the level by right-clicking in the viewport somewhere within the camera frustum, and selecting Create entity.
Creating Multiple Bundles

b. In the Asset Browser view, search for the `am_rock_boulder_01.cgf` mesh.

![Asset Browser](image)

4. Make sure that the boulder is visible from the camera position by running your game in the editor. Select Game > Play Game (Ctrl-G). If you can't see the boulder, adjust its position with the editor's Move tool and make sure it appears within the camera's view.

5. Select File > Save (Ctrl-S) to save the level.

6. Select Game > Export to Engine (Ctrl-E) to export the level.

7. Exit the editor.

Generate the new content bundle

In the Build and bundle assets for release in Lumberyard (p. 344) tutorial, you created two `.pak` bundles for the game release. This new level you've created only uses the already-available assets that
are bundled with your game, making it easy to distribute only the content that you need. One important feature available in Lumberyard Beta v1.24 and later is the ability to distribute level data as part of its own .pak, rather than with the auxiliary game data.

This section of the tutorial walks you through creating a new asset list based on the level2.pak dependencies, removing duplicate entries that already appear in startergame_pc.pak, and bundling the level for distribution.

1. Create the seed list and asset list for level2.pak:

```
Bin64vc141\AssetBundlerBatch.exe seeds --addSeed levels\level2\level.pak --seedListFile dlc_level2.seed
Bin64vc141\AssetBundlerBatch.exe assetLists --seedListFile dlc_level2.seed --assetListFile dlc_level2_all.assetlist
```

2. Get the complement of the assets between startergame_pc.assetlist from the previous tutorial and dlc_level2_all_pc.assetlist. This generates a new asset list containing game content exclusive to level2.

```
Bin64vc141\AssetBundlerBatch.exe compare --comparisonType complement ^
   --firstAssetFile startergame_pc.assetlist ^
   --secondAssetFile dlc_level2_all_pc.assetlist ^
   --output dlc_level2.assetlist
```

**Important**
The ordering of `--firstAssetFile` and `--secondAssetFile` here is required. The compliment comparison works by taking content located in `secondAssetFile` which isn't referenced in `firstAssetFile` — not the other way around. See Amazon Lumberyard Asset List Comparison Operations (p. 380) for all of the details.

3. Get the complement of the assets between engine_pc.assetlist from the previous tutorial and dlc_level2_pc.assetlist. This removes engine-specific content from the level2 assets.

```
Bin64vc141\AssetBundlerBatch.exe compare --comparisonType complement ^
   --firstAssetFile engine_pc.assetlist ^
   --secondAssetFile dlc_level2_pc.assetlist ^
   --output dlc_level2.assetlist ^
   --allowOverwrites
```

4. Create the asset bundle for level2:

```
Bin64vc141\AssetBundlerBatch.exe bundles --assetListFile dlc_level2_pc.assetlist --outputBundlePath %USERPROFILE%\StarterGameRelease\startergame\dlc_level2.pak
```

**Test the new level in release**

1. Open a console and navigate to the release directory at %USERPROFILE%\StarterGameRelease
2. Launch the new map:

```
release\StarterGameLauncher.exe +map level2
```

If everything works, you should see a scene similar to the following:
Conclusion

You now have a working example of a release build of a PC game on Lumberyard that uses multiple asset bundles. These bundles do not contain duplicate files, and you've performed a workflow that matches how a downloadable content system would add new content to an already released game.

Next Steps

Learn how to bundle patch updates instead of all-new content with the next tutorial in this series: Create content patches with Lumberyard (p. 354).

Create content patches with Lumberyard

This tutorial is out of date for the new Starter Game project that shipped as part of Lumberyard 1.25. We're working to revise it and provide better, more comprehensive instructions.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This tutorial demonstrates a bundling process designed to simulate a patch update to existing content. In this tutorial, you create a set of bundles that simulates a content patch. These bundles include the patch itself, and a new version of the software to distribute with the patch already in place. The update bundle contains only content that changed since the first set of bundles was created. This topic covers:

- Modifying an asset reference to reference a new asset that was not bundled previously.
- Creating an updated asset bundle that contains only updated or new assets.
- Running the new release build with new bundles.
Prerequisites

To complete this tutorial, you should have completed the Build and bundle assets for release in Lumberyard (p. 344) and Creating Multiple Asset Bundles (p. 349) tutorials. You should have:

- A bundled release of Starter Game located at `%USERPROFILE%\StarterGameRelease`
- The startergame_pc.assetlist file used to generate the starter game content bundle.
- The dlc_level2.seed seed list file used to generate the asset list for the level content bundle.
- The dlc_level2_pc.assetlist file used to generate the additional level content bundle.

Add a gem and modify the level

For this tutorial you'll modify the level2 level created in the previous tutorial. You'll also be adding a new asset-only gem to the Starter Game project, to demonstrate how to update auxiliary content in addition to pushing direct changes to a bundle.

1. Launch the Project Configurator, make sure that the Starter Game is selected, and select Enable Gems.

![Lumberyard Project Configurator (PREVIEW)](image)

![Select a project](image)

2. Search for the term primitive in the search bar. The Primitive Assets gem should appear in the results list. Select the checkbox to include it as part of the Starter Game, and then select Save.
Create content patch bundles

Note

The Primitive Assets gem is an asset-only gem, so you aren't required to recompile any code for your game. If you were adding a gem with a code component, you would also need to rebuild both the profile and release portions of your game and distribute the new binaries as part of the patch. See the Build and bundle assets for release in Lumberyard (p. 344) for build instructions and where the release binaries need to be located.

3. Open the Lumberyard Editor and then open level2 that you created in the previous tutorial. (p. 349)

4. Create a new entity in the level and assign it the _box_1x1.fbx mesh.
   a. Right-click in the viewport, somewhere within the frustum of the camera, and select Create entity.
   b. Search for _box_1x1.fbx in the Asset Browser.
   c. Drag the _box_1x1.fbx mesh onto your newly created entity in the level.

If you need a visual refresher on the editor UI for these steps, see Creating Multiple Asset Bundles (p. 349).

5. Save the level by selecting File > Save (Ctrl-S) from the main menu.

6. Export the level by selecting Game > Export to engine (Ctrl-E) from the main menu.

7. Close the Lumberyard Editor.

Recreate the project bundles

In this step you'll version your old asset lists, generate new asset lists based on the changes to the level, and create two content packs: One for distributing to existing versions of the game as a patch, and one for directly downloading the latest version.

Note

For most game updates, only the content of your game assets will change. When upgrading to a new version of Lumberyard and pushing an update, you should regenerate all of the existing content to ensure it's properly updated, including the engine assets and auxiliary content.

1. Open a console and navigate to the Lumberyard root directory at lumberyard_dir\dev.

2. Add a version identifier to the existing dlc_level2_all_pc.assetlist asset list:

   move dlc_level2_all_pc.assetlist dlc_level2_v1_all_pc.assetlist

3. Generate a new asset list dlc_level2_v2_all_pc.assetlist from the dlc_level2.seed file:

   Bin64vc\AssetBundlerBatch.exe assetLists --seedListFile dlc_level2.seed --assetListFile dlc_level2_v2_all.assetlist

   Note

   You don't need to update the dlc_level2.seed file here because only the dependences of the level have changed - nothing about the asset itself that's used as the seed.

4. Get the delta of the assets between the v1 and v2 contents. Taking a delta gets everything that's changed between the two versions and generates a patch:

   Bin64vc\AssetBundlerBatch.exe compare --comparisonType delta ^
   --firstAssetFile dlc_level2_v1_all_pc.assetlist ^
   --secondAssetFile dlc_level2_v2_all_pc.assetlist ^
Create content patch bundles

--output dlc_level2_v1_to_v2_patch.assetlist

5. Remove starter game and engine assets from the dlc_level2_v2_all_pc.assetlist and generate a new asset list with only the latest required level2 assets:

```bash
Bin64vc\AssetBundlerBatch.exe compare --comparisonType complement ^
--firstAssetFile startergame_pc.assetlist ^
--secondAssetFile dlc_level2_v2_all_pc.assetlist ^
--output dlc_level2_v2.assetlist
```

```bash
Bin64vc\AssetBundlerBatch.exe compare --comparisonType complement ^
--firstAssetFile engine_pc.assetlist ^
--secondAssetFile dlc_level2_v2_pc.assetlist ^
--output dlc_level2_v2.assetlist ^
--allowOverwrites
```

6. Create two new bundles, one for the patch and one for the new full distribution:

```bash
Bin64vc\AssetBundlerBatch.exe bundles --assetListFile dlc_level2_v1_to_v2_patch.assetlist --outputBundlePath dlc_level2_v1_to_v2_patch.pak
```

```bash
Bin64vc\AssetBundlerBatch.exe bundles --assetListFile dlc_level2_v2_pc.assetlist --outputBundlePath dlc_level2_v2.pak
```

**Important**

Don't copy either of these files to the release folder yet. The next section will cover testing each bundle to make sure that it contains the correct content, and that they load correctly.

## Simulate user scenarios

Now you're ready to simulate two scenarios: users who have v1 of your game but upgrade to v2, and users who download v2 for the first time.

### Simulate the v1 to v2 patch experience

1. Open a console and navigate to the Lumberyard root directory.
2. Copy the content patch to your release:

```bash
copy dlc_level2_v1_to_v2_patch_pc.pak %USERPROFILE%\StarterGameRelease\startergame
```
3. Launch the starter game and check to see if the new 1x1 box appears in the level2 scene.

```bash
%USERPROFILE%\StarterGameRelease\release\StarterGameLauncher.exe +map level2
```

If the content patch was applied successfully, you should see a scene similar to the following:
Simulate the v2 download experience

1. Delete the content patch and v1 data from your game release:

   ```
   del %USERPROFILE%\StarterGameRelease\startergame\dlc_level2_v1_to_v2_patch_pc.pak
   del %USERPROFILE%\StarterGameRelease\startergame\dlc_level2_pc.pak
   ```

2. Copy the level2 v2 content to the release:

   ```
   copy dlc_level2_v2_pc.pak %USERPROFILE%\StarterGameRelease\startergame
   ```

3. Launch the starter game and check to see if the new 1x1 box appears in the level2 scene.

   ```
   %USERPROFILE%\StarterGameRelease\release\StarterGameLauncher.exe +map level2
   ```

**Conclusion**

You've learned how to generate a content patch to apply to an existing game. Now you know the general steps for distributing additional content and patches for your Lumberyard game:

- Create a complete asset list using your newly created or updated content as a seed.
- Take the *complement* of this asset list from the original game assets and engine assets, to get only the assets needed by your new content.
- When you patch existing content, take the *delta* of the patch asset list against the asset list for the content you're applying the patch to.
Migrating a Game Project to the Asset Bundler

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you already have an existing Lumberyard project, and you want to use the new Asset Bundler for your game assets, complete the following steps to update your project:

- Fix missing product dependencies.
- Create a release build using files that are generated with the Asset Bundler.

Fix Missing Dependencies

The Asset Bundler relies on accurate and complete product dependency information to ensure that all assets that are needed for runtime are included in the asset bundle. This procedure shows how to use new and existing Lumberyard tools to find and fix missing product dependencies.

1. Identify missing product dependencies in your project. The missing dependency scanner tool is useful for finding missing product dependencies in projects that don't currently use asset bundling.
   - Using the Missing Dependency Scanner (p. 366)
   - Resolving Missing Assets (p. 363)
2. Update your custom asset builders to output product dependencies. Be sure to add false positive results to the exclusion list using the file tagging system.
   - Implement the Callback for ProcessJobFunction (p. 322)
   - Using the File Tagging System to Include or Exclude Assets (p. 377)
3. Create a seed list and add your required assets as seeds. You can create a dependencies XML file (such as Check Engine/Engine_Dependencies.xml) to include optional assets or add multiple seeds using wildcards. Include the dependencies XML file in the seed list.
   - What is the Amazon Lumberyard Asset Bundler? (p. 335)
4. After you add missing product dependencies and create seed lists, check that product dependencies are being emitted correctly using the Asset Validation Gem with seed mode enabled.
   - Using the Asset Validation Gem to Verify Seeds (p. 371)
5. Generate asset bundles using the Asset Bundler command line tools.
   - Build and bundle assets for release in Lumberyard (p. 344)
   - Lumberyard Asset Bundler Command-Line Tool Reference (p. 394)
6. Have the system load up the bundles that you created in Step 5. Then, use bundle mode to identify any missing assets in the bundle. Ideally, you don't want to see any missing dependencies, but there is no requirement to resolve them all.
   - Using Bundle Mode to Test Bundles (p. 373)

Create Release Build with New Asset Bundling System

After you generate asset bundles to deliver with your game, switch your release build process to use the new asset bundling system. Use the following instructions to generate content including asset bundles for a release build.
1. Configure your game project and set it as the active project. Go to the starter game walkthrough (Build and bundle assets for release in Lumberyard (p. 344)) and follow Steps 1 through 4. Be sure to specify your own game project name instead of "StartGame." The instructions provided in the walkthrough are for PC builds. If your game uses another operating system, such as macOS, iOS, or Android, specify the correct profile in your lmbr_waf and lmbr_pak_shaders calls.

2. If your game runs on the PC operating system, you are ready to create the release build for your game. You can do this by using command line calls, as described in Steps 5 through 6 of the starter game walkthrough. Alternatively, create custom scripts to generate your asset bundles and make .pak files for your content. You can customize the samples described in the next section.

3. If your game runs on any operating system other than PC, follow these steps:
   a. Create a custom RC config file. You can use the file RCJob_Generic_MakeAuxiliaryContentPC.xml as a template and update the operating system and any OS-specific settings.
   b. Create a customized batch file. You can use the file Build_AssetBundler_AuxiliaryContent_PC.bat as a template and update the operating system. You also must update the reference to the RC config file to point to the file that you created in the previous step.
   c. Run the customized batch file to generate auxiliary data and the asset bundles.
   d. Add the new bundle files to the directory gamename_platform_paks/gamename. You can do this manually or create custom scripts based on the samples described in the next section.

Release Build Sample Scripts

The following sample scripts show how you can simplify and automate basic release build processes for your game projects.

- generate_bundle.py (p. 360) – This script uses the Lumberyard command line calls to generate asset bundle files.
- ProjectName_MakePaks.py (p. 362) – This script creates a folder titled gamename_pc_paks and copies the base bundle files to the subfolder gamename_platform_paks/gamename.

```python
import argparse
import subprocess
import os
import platform

class Args(object):
    '''An object that returns None for any undefined properties.'''

    def __init__(self, **kwargs):
        if kwargs:
            for k, v in kwargs.items():
                self.__dict__[k] = v

    def __str__(self):
        return str(self.__dict__)

    def __getattr__(self, name):
        if name in self.__dict__:
            return self.__dict__[name]
        else:
            return None
```

Version 1.28

360
def generate_bundle(args):
    pak_name = args.level
    seed_list_file = 'ProjectName/SeedAssetList.seed'
    asset_list_path = os.path.join('ProjectName', 'AssetLists', pak_name + '_' +
                                args.platform + '.assetlist')
    bundle_path = os.path.join('ProjectName', 'Bundles', pak_name + '_' + args.platform +
                                '.pak')

    if platform.system() == 'Windows':
        asset_bundler_batch = 'Bin64vc141/AssetBundlerBatch.exe'
    elif platform.system() == 'Darwin':
        asset_bundler_batch = './BinMac64/AssetBundlerBatch'

    asset_lists_command = [
        asset_bundler_batch,
        'assetLists',
        '--assetListFile={0}'.format(asset_list_path),
        '--platform={0}'.format(args.platform),
        '--allowOverwrites'
    ]

    if args.level:
        seed = 'levels'
        seed = os.path.join(seed, args.level, 'level.pak')
        asset_lists_command.append('--addSeed={0}'.format(seed))

    if args.base:
        asset_lists_command.append('--seedListFile={0}'.format(seed_list_file))
    asset_lists_command.append('--addDefaultSeedListFiles')
    if args.seedListFile:
        asset_lists_command.append('--seedListFile={0}'.format(args.seedListFile))

    # Generate the asset list file
    subprocess.check_call(asset_lists_command)

    # Generate the bundle file
    subprocess.check_call(  
        [asset_bundler_batch,  
         'bundles',  
         '--assetListFile={0}'.format(asset_list_path),  
         '--outputBundlePath={0}'.format(bundle_path),
         '--platform={0}'.format(args.platform),
         '--allowOverwrites']
    )

    print("Asset list file is stored at {}".format(asset_list_path))

    expected_bundle_path = os.path.join('ProjectName', 'Bundles', pak_name +
                                         '.' + args.platform + '.pak')
    if os.path.exists(expected_bundle_path):
        os.remove(expected_bundle_path)
    os.rename(bundle_path, expected_bundle_path)
    print("Bundle file is stored at {}".format(expected_bundle_path))

parser = argparse.ArgumentParser()
parser.add_argument('--level', help='Specify the level name')
parser.add_argument('--platform', required=True, help='Specifies the platform that will be referenced when generating Bundles.')
parser.add_argument('--base', action='store_true', help='Specifies whether it is the base level.')
parser.add_argument('--seedListFile', help='Specifies the seed list file(s) for the level.')
parser.set_defaults(func=generate_bundle)
import argparse
import subprocess
import os
import sys
import shutil

def main(context, platformList):
    for platform in platformList:
        # run MakePaks batch (i.e. BuildSamplesProject_Paks_PC.bat) to make base install paks
        subprocess.check_call([context['PakScript'],
                               context['PlatformArgFormat'].format(platform)])
        # run GenerateBundle.py script to generate a level pak
        subprocess.check_call([context['GenerateBundleCmd'] + ['--platform', platform]])
        # copy the bundle from ProjectName/Bundles/levelName.<platform>.pak into Cache/
        shutil.copyfile(os.path.join(context['BundleFolder'], 'levelName.{}.pak'.format(platform)),
                         os.path.join(context['CachePath'], '{}_paks'.format(platform), 'projectname',
                                      'levelName.{}.pak'.format(platform)))

debug_dir = os.path.dirname(os.path.realpath(__file__))
context = {
    'PakScript': '',
    'GenerateBundleCmd': '',
    'BundleFolder': os.path.join(debug_dir, 'ProjectName', 'Bundles'),
    'CachePath': os.path.join(debug_dir, 'Cache', 'ProjectName'),
    'PlatformArgFormat': ''
}

# Run this to make the install paks for your app
mygame_pak_script = 'ProjectName_MakePaks.{}
python = ''

if sys.platform == 'darwin':
    context['PakScript'] = os.path.join(debug_dir, mygame_pak_script.format('sh'))
    python = 'python'
    context['PlatformArgFormat'] = '-p={}
else:
    context['PakScript'] = os.path.join(debug_dir, mygame_pak_script.format('bat'))
    python = os.path.join(debug_dir, 'Tools', 'Python', 'python.cmd')
    context['PlatformArgFormat'] = '{}

context['GenerateBundleCmd'] = [python, os.path.join(debug_dir, 'generate_bundle.py'), '--level', 'levelName', '--base']

parser = argparse.ArgumentParser()
parser.add_argument('--platform', help='Specifies the platform that will be referenced when generating Bundles.')(args = parser.parse_args())

print(args)
main(context, [args.platform])
Resolving Missing Assets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you build and package your Lumberyard game, you want to frequently verify that your packages contain every asset they require. For information about verifying your asset bundles, see Verifying that Bundles Contain Required Assets (p. 365).

When you identify a potential missing asset, you want to include it so that the asset is no longer missing in your next bundled game package.

An asset that is missing from a bundle might be one of the following:

- **A missing product dependency** – An asset that references another asset, but did not declare it as a product dependency, or the referenced asset was removed during the asset list comparison process (p. 380).
- **A hardcoded file load** – Assets loaded by path or by asset ID in C++.
- **A false positive** – The asset appeared to be missing from a bundle, but is not actually used. For example, you might have an editor-only asset that appears to be missing that is never loaded or used in your game’s launcher.

To resolve the missing asset, check each of these possibilities in turn.

### Missing Product Dependencies

A missing asset might have been loaded as a reference from Lumberyard or from your game code's interaction with another asset. In these cases, you can resolve the issue by emitting a new product dependency.

#### Finding the Asset Reference

To find the source of the asset reference, try the following approaches:

- Use the Asset Processor Batch's missing dependency scanner (p. 366).
- Debug the file load using the following methods:
  - Set breakpoints, if possible
  - Add extra print commands
  - Try interacting with your game in different ways to see what triggers the missing asset to load.

As a hint to where to start debugging, note that most file loading routes through `lumberyard_version\dev\Code\CryEngine\CrySystem\CryPak.cpp` in `CCryPak::FOpen` and `CCryPak::GetFileData`. Setting breakpoints or adding additional debugging logic (like printf statements) can help identify what is triggering the asset load that you are investigating.

#### Finding the Builder to Update

Track down the job that generates the asset that references your missing asset. Then you can update the generated asset to emit the missing asset as a product dependency. The source of the generated asset can be obvious if it has a file extension that is associated with a known builder.

If the source of the generated asset is not obvious, you can use the asset database to look up the information.
To look up a job that generates an asset

1. In the **Products** table, search for the asset that you want to update to emit a dependency. The following example search for a material uses **DB Browser for SQLite** to explore the asset database:

![DB Browser for SQLite screenshot](image1.png)

2. Look up the **JobPK** value in the **Jobs** table, as in the following example:

![DB Browser for SQLite screenshot](image2.png)

3. Look up the **JobKey** or **BuilderGuid** in your code base.

**Updating the Builder to Emit the Dependency**

After you've identified the builder which emits the product that is missing the dependency, update the builder. For more information, see Declare Product Dependencies (p. 322).

**Hardcoded File Loads**

To resolve missing assets from hardcoded file loads, find where in code the file is loaded and then choose a resolution strategy.

**Finding the File Load**

We recommend the following techniques for tracking down a hardcoded file load:

1. Using breakpoints in code.
2. Adding additional log messages to the code base.
3. Doing a “find in files” search for strings that refer to the missing asset.

For more information, see Finding the Asset Reference (p. 363) earlier in this topic.

**Resolving the Missing Asset**

To resolve the missing asset from a hardcoded file load, try the following options:

- **Remove the hardcoded load** – By emitting assets as product dependencies from relevant builders, you can use seed lists with fewer files that are easier to maintain.
- **Add as seed** – If you can't or don't want to replace the hard-coded asset load, you can add the referenced file as a seed to your game's seed list. Because adding the seed changes only data and
doesn't require recompiling your game, this approach can be useful later in development and 
minimizes code changes. For information about adding the referenced file as a seed to your game's 
seed list, see the Lumberyard Asset Bundler Command-Line Tool Reference (p. 394).

• **Use the Wildcard Dependency System** – If your project uses relative path loads or wildcard path 
loads, you can declare the dependencies in a dependencies file. This technique is explained in the 
following section.

### Using the Wildcard Dependency System to Resolve Path Loads

When you migrate (p. 359) a Lumberyard project to use seeds or emit dependencies for all referenced 
assets, two cases cannot be resolved as seeds: Optional relative path loads and wildcard path loads. If 
your bundle is missing an asset that is loaded in either style, try to resolve the missing asset by using 
wildcard dependencies.

To handle dependency tracking for runtime systems that use path manipulation or directory 
scanning to load product files, declare dependencies in a `_Dependencies.xml` file. For example, 
dependencies in the core engine are included in the `lumberyard_version\dev\Engine 
\Engine_Dependencies.xml` file. A dependencies file uses the following format.

```xml
<EngineDependencies versionnumber="1.0.0">
  <Dependency path="*.ent" />
</EngineDependencies>
```

The `path` value is a relative path to a product file that uses simple glob searching. Asterisk (*) characters 
match any number of characters. Glob searches do not support single character wildcards. The files listed 
are recorded as dependencies.

After you add a new `_Dependencies.xml` file, add an entry for the referenced file to a seed list to 
ensure full dependency tracking. The core `Engine_Dependencies.xml` file is already included in the 
default seed list.

### False Positives

Some assets and asset references are used only in the editor or in launchers during development. These 
assets aren't used in release builds. Therefore, you can consider any assets that are missing from bundles 
that aren't used in release builds to be false positives.

### Removing False Positives From Missing Asset Scanning Results

After you've verified that an asset is not used in your release builds, you can use the file tagging system 
to tag it so that it doesn't appear in future scans. For more information, see Using the File Tagging 
System to Include or Exclude Assets (p. 377).

### Verifying that Bundles Contain Required Assets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
Download O3DE or visit the AWS Game Tech blog to learn more.

When your team generates a release build that has bundled game assets, you want to verify that the 
bundles contain everything required to ship your game. Use the tools of the Lumberyard asset bundling 
system throughout the game development process to verify that your asset bundles contain the required 
assets.

As a best practice, verify that your bundles contain their required assets during all game development 
phases:
1. When you start
2. While you develop
3. In development builds
4. In release builds

**When You Start**

As you start building your game, ensure that you create the references to assets that your product requires at runtime. For information about how to emit product dependencies for your custom asset types, see the Declare Product Dependencies (p. 322) section of the Creating a Custom Asset Builder (p. 318) page.

**While You Develop Your Game**

Errors can occur when assets are bundled later in the development process. To avoid errors, consider bundling earlier. However, even before you've defined your seed list or built bundles, you can take steps to avoid unwanted surprises later.

**Before You Build a Seed List**

Before you build a seed list, you can use the missing dependency scanner to look for asset references that might have missing dependencies. For more information, see Using the Missing Dependency Scanner (p. 366).

**After You Have a Seed List**

After you've built a seed list, but before you bundle your assets, you can use the Asset Validation Gem to verify that asset loads map back to seeds. For more information, see Using the Asset Validation Gem to Verify Seeds (p. 371).

**In Development Builds**

After you've generated bundles for your game, test your game in bundle mode with a development build of the editor or launcher. With bundle mode active, use the `sys_report_files_not_found_in_paks` console variable to find any asset files that are missing from your game .pak files. For more information, see Using Bundle Mode to Test Bundles (p. 373).

**In Release Builds**

As with development builds, you can use bundle mode and the `sys_report_files_not_found_in_paks` console variable to find missing asset files in release builds. For more information, see Using Bundle Mode to Test Bundles (p. 373).

**Topics**

- Using the Missing Dependency Scanner (p. 366)
- Using the Asset Validation Gem to Verify Seeds (p. 371)
- Using Bundle Mode to Test Bundles (p. 373)

**Using the Missing Dependency Scanner**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Use the **AssetProcessorBatch.exe** tool to scan your files for patterns that look like missing dependencies. To perform a scan, run the **AssetProcessorBatch.exe** command using the `dependencyScanPattern` (or `dsp`) flag. Because the command performs a SQL query, use search strings compatible with SQL syntax; for example, the wildcard character is `%`, not `*`.

The following command scans all files with the extension `.txt`.

```
AssetProcessorBatch.exe /dependencyScanPattern=%txt
```

The command scans the **ProductName** column in the **Products** table in the asset database. The scanner performs multiple passes so that it can present only results that are likely to be missing dependencies. It scans for patterns like paths, asset IDs, and UUIDs that map to existing products in your game. The results of the scan are stored in the **MissingProductDependencies** table of the asset database.

**Taking Action**

If a scan of assets in your project returns results, verify that they are true dependencies.

- If the results are false positives, use the asset tagging system to mark the files that can be safely ignored in future scans. Files tagged with the `editoronly` or `shader` flags in the file `exclude.filetag` are skipped by the scanner. For more information about file tagging, see Using the File Tagging System to Include or Exclude Assets (p. 377).
- If the dependencies reported are truly missing, update the builder for your asset to emit a dependency. For more information, see the Declare Product Dependencies (p. 322) section of the Creating a Custom Asset Builder (p. 318) page.

**Example**

The following procedures show you how to create an XML file that has a missing dependency. You then resolve the dependency by creating a new schema.

**To create a file with a missing dependency**

1. Create a new XML file in your project named `missingdependency.xml`.
2. Edit the file to include a reference to an asset in your project. In the following example, `missingDependency` refers to the `project.json` file in the game project.

   ```xml
   <missingDependency path="project.json"/>
   ````

3. Run the missing dependency scanner, which processes assets, using syntax, as shown in the following example. The command uses a wildcard to avoid including a platform or project name in the query.

   ```
   Bin64\vc141\AssetProcessorBatch /dsp=%missingdependency.xml
   ```

   The following example shows a query without a wildcard character. The query matches the sample project and operating system environment. Replace `<project_name>` with the name of your game project.

   ```
   Bin64\vc141\AssetProcessorBatch /dsp=pc/<project_name>/missingdependency.xml
   ```

4. Observe the log output. The UUID in the following sample might be different from what you see.

   ```
   AssetProcessor: Scanning for missing dependencies: I:\p4\lyengine\branches \<project_name>\dev\Cache\<project_name>\pc\<project_name>\missingdependency.xml
   ```
AssetProcessor: Missing dependency: String "project.json" matches asset: 
{B076CDDC-14DF-50F4-A5E9-7518ABB3E851}:0

The following image shows sample asset database output.

Next, create a schema to match this file and output, or emit, the dependency.

**To create a schema to emit the dependency**

1. In Lumberyard Editor, choose Tools, Asset Editor.
2. In the Asset Editor, choose File, New, XML Schema.
3. For Matching Rules, click the plus (+) icon to add a rule.
4. For File Path Pattern, enter missingdependency.xml.

Version 1.28
368
5. For **Dependency Search Rules**, add a dependency search rule and a search definition.
6. For **Search Rule Structure, Name**, enter the name `missingDependency`.
7. For **Attributes**, add an attribute with the following values.
   a. For **Name**, enter `path`.
   b. For **Expected Extension**, enter `.json`.
   c. For **Path Dependency Type**, choose `ProductFile`.
8. Save the file to the location `lumberyard_version\dev\Engine\Schema\missingdependency.xmlschema`.
9. Save the `missingdependency.xml` again to force it to reprocess.
10. Verify that the dependency now appears correctly in the asset database, as the following images show:
11. Run the missing dependency scanner again, using syntax, as shown in the following example:

```
Bin64vc14\AssetProcessorBatch /dsp=%missingdependency.xml
```

12. The scanner should now report no results, as shown in the following sample log entry:

```
AssetProcessor: Scanning for missing dependencies: I:\p4\lyengine\branches \<project_name>\dev\Cache\<project_name>\pc\<project_name>\missingdependency.xml
```
Custom Dependency Scanners

The missing dependency scanning system supports authoring specialized dependency scanners. To match specific data types, you can build a custom scanner by inheriting from `SpecializedDependencyScanner`. Implement the necessary functions and register your scanner with the `MissingDependencyScanner` by calling `RegisterSpecializedScanner`.

You can specify which scanners are run against which files by changing the call to `MissingDependencyScanner`'s `ScanFile` to use a different match type or force the usage of a particular scanner.

Using the Asset Validation Gem to Verify Seeds

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you've built a seed list but before you bundle, you can use the Asset Validation gem to verify that asset loads map back to seeds. The Asset Validation gem adds a set of seed-related commands to the Lumberyard console command window. You can use these commands to ensure that you have seeds for all assets that you want to bundle.

One of these commands, `seedmode`, makes a seed mode active. When seed mode is active and an asset file loads, Lumberyard traverses the dependency graph from the newly loaded asset until it finds a seed for the asset. If it doesn't find a seed for the asset, an error message displays. It warns you that the loaded file won't be bundled if you try to bundle using the seeds that you provided for the current validation session.

During development, use seed mode to ensure that as assets get added they're properly bundled and included in the shipping version of the game.

**Note**

If you already have bundles to test, you can use `bundle mode` instead of seed mode. For more information, see Using Bundle Mode to Test Bundles (p. 373).

Prerequisites

**Enable** (p. 1064) the Asset Validation gem in your game project, and then **build** (p. 61) your game project.

Seed Mode Commands

When you use Lumberyard Editor or the launcher to run your game, the following console commands are available:

- `seedmode` – Enables or disables the reporting system.
- `addseedpath <Relative cache path to an asset>` – Adds the specified asset and all of its dependencies as a seed to your dependency graph, so they no longer are reported as missing. The `addseedpath` command lets you test which assets should be added to your seed list for packaging.

**Example**

AssetProcessor: No assets remain in the build queue. Saving the catalog, and then shutting down.
Verifying Bundles

- **addseedpath** `<Relative cache path to an asset>` – Removes an asset from the dependency graph.
- **removeseedpath** `<Relative cache path to an asset>` – Removes an asset from the dependency graph.
- **listknownassets** – Lists all the assets in your current dependency graph.
- **addseedlist** `<Relative source path to a seed list>` – Adds all the seeds found in the specified seed list to your known dependency graph. Because the path is a source path, it is relative to the `lumberyard_version\dev` directory.

Example

```
addseedlist Engine\SeedAssetList.seed
```

- **removeseedlist** `<Relative source path to a seed list>` – Removes all the assets in a seed list from your graph.
- **printblacklisted** – Enables or disables the display of approved assets in the system. Some assets, like shaders, are loaded at runtime and do not appear in your dependency graph. By design, shaders are packaged in their own `.pak` file, are not found in the dependency graph, and do not need to be reported by the system. However, you can use the `printblacklisted` command to force shaders or other approved asset types to be included in the dependency graph.

Using Seed Mode

The following procedure shows how to use seed mode to troubleshoot a level that has missing assets.

**To test seed mode**

1. In the Lumberyard console, enter the command `seedmode`. Asset validation begins.

   ```
   seedmode
   [CONSOLE] Executing console command 'seedmode'
   (AssetValidation) - Asset Validation is now on
   ```

2. Enter game mode. In the console, seed mode reports multiple `Asset not found in seed graph` errors.

3. Exit game mode.

4. Enter the `addseedpath` command to add the missing asset file. This example uses the command `addseedpath levels\milestone2\level.pak`.

5. Enter game mode. The `Asset not found` errors no longer appear.
Handling Missing Asset Errors

If seed mode reports that an asset is missing, the asset might be one of the following:

- Part of a list that you haven't added to the graph yet.
- An asset that must be added as a dependency of another asset already found in the level.
- An asset that must be a seed itself.

For more information, see Find and Fix Missing Asset References.

Using Bundle Mode to Test Bundles

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Bundle mode is a process that lets you enable asset loading to prioritize bundles over loose cache assets. After you build the seed lists for packaging your game, you can use bundle mode and the `sys_report_files_not_found_in_paks` console variable to test your packaging rules. Bundle mode makes it easy for you to load and report on issues in all the bundles (game .pak files) from a location that you specify without creating a release build.

Using bundle mode involves two key tasks:

- Turning on missing asset reporting for assets not in bundles. This enables "bundle mode".
- Mounting and loading bundles for your game.

When reporting is enabled, the `sys_report_files_not_found_in_paks` console variable reports when an asset loads that isn't in any of your bundles. By selectively loading bundles and using the `sys_report_files_not_found_in_paks` command, you can find assets that need to be included in your bundles.

Enabling Bundle Mode

To enable bundle mode, use the `sys_report_files_not_found_in_paks` console variable and specify a value of 1, 2, or 3. A value of 1 writes missing files as log entries without issuing warning messages.
The following list shows valid arguments for the `sys_report_files_not_found_in_paks` console variable.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disabled</td>
</tr>
<tr>
<td>1</td>
<td>Log</td>
</tr>
<tr>
<td>2</td>
<td>Warning</td>
</tr>
<tr>
<td>3</td>
<td>Error</td>
</tr>
<tr>
<td>&lt;Every other value&gt;</td>
<td>Error</td>
</tr>
</tbody>
</table>

Log File Entries

Missing files are logged with entries similar to the following:

```
Missing from bundle: @assets\levels\milestone2\auto_resourcelist.txt
```

If you use the `sys_report_files_not_found_in_paks` console variable with the launcher, the error messages are written to a log file in your cache user directory. On a PC, these are located at `lumberyard_version\dev\Cache\project_name\pc\user\log\logs\pakmissingassets.log`.

Setting the Console Variable

Enabling the console variable before running the editor or launcher ensures that all missing assets are reported. To ensure that the console variable is always active when you run the editor or launcher, modify the following files:

Editor: `lumberyard_version\dev\editor.cfg`

Runtime: `lumberyard_version\dev\project_name\autoexec.cfg`

You can also enable the console variable at runtime by using the console (~) or remote console. For information about the remote console, see Universal Remote Console (p. 3209).

Bundle Mode Commands

Bundle mode has two commands:

- `loadbundles <bundle_directory> <extension>` – Loads all the bundles from the specified directory into the game. If no arguments are supplied, the directory defaults to Bundles and the extension to .pak.

  For example, if you run the Starter Game project and enter the command `loadbundles` without arguments, Lumberyard loads all source files in the `lumberyard_version\dev\StarterGame\Bundles` directory that have the extension .pak.

- `unloadbundles` – Unload any bundle that was loaded through the `loadbundles` command.

Using Bundle Mode Example

The following procedure shows how bundle mode works. In the example, game mode is entered when a bundle is missing.

To test bundle mode

1. In the console window, enter the following command:

```
sys_report_files_not_found_in_paks 1
```
The 1 argument specifies that missing files are reported as log entries rather than warnings or errors.

2. Enter game mode. A list of Missing from bundle errors displays.

```
[Warning] (Cloud Canvas) [Cloud Canvas] Resource manager was invoked but there are no deployments or resource groups.
Disable Accelerators
(CryPak) - Missing from bundle: @assets\scriptcanvas\move\tank\scriptcanvas\compiled
(CryPak) - Missing from bundle: @assets\scriptcanvas\look\scriptcanvas\compiled
(CryPak) - Missing from bundle: @assets\scriptcanvas\camera\scriptcanvas\compiled
(CryPak) - Missing from bundle: @assets\ui\milestone2\menu\ui\canvas
(CryPak) - Missing from bundle: @assets\ui\milestone2\menu\ui\canvas
(CryPak) - Missing from bundle: @assets\levels\milestone2\hardcode\assetref\luac
(CryPak) - Missing from bundle: @assets\ui\textures\prefab\button\disabled\sprite
(CryPak) - Missing from bundle: @assets\ui\textures\prefab\button\normal\sprite
(CryPak) - Missing from bundle: @assets\levels\milestone2\milestonescene\scriptevents
(CryPak) - Missing from bundle: @assets\levels\milestone2\milestonescene\scriptevents
(CryPak) - Missing from bundle: @assets\scriptcanvas\milestone2\button\scriptcanvas\compiled
(CryPak) - Missing from bundle: @assets\levels\milestone2\milestonescene\scriptevents
(CryPak) - Missing from bundle: @assets\levels\milestone2\milestonescene\scriptevents
(CryPak) - Missing from bundle: @assets\levels\milestone2\milestonescene\scriptevents
```

3. Enter the command `loadbundles` to load bundles for the level.

```
loadbundles

[CONSOLE] Executing console command 'loadbundles'
<table>
<thead>
<tr>
<th>BundlingSystem</th>
<th>Loading bundles from bundles of type .pak</th>
</tr>
</thead>
<tbody>
<tr>
<td>BundlingSystem</td>
<td>Opening bundle @assets\bundle\engine\dependencies.pak</td>
</tr>
<tr>
<td>BundlingSystem</td>
<td>Loaded bundle @assets\bundle\engine\dependencies.pak</td>
</tr>
<tr>
<td>BundlingSystem</td>
<td>Opening bundle @assets\bundle\milestone2.pak</td>
</tr>
<tr>
<td>BundlingSystem</td>
<td>Loaded bundle @assets\bundle\milestone2.pak</td>
</tr>
<tr>
<td>BundlingSystem</td>
<td>Opening bundle @assets\bundle\spying.pak</td>
</tr>
<tr>
<td>BundlingSystem</td>
<td>Loaded bundle @assets\bundle\spying.pak</td>
</tr>
<tr>
<td>general.enter_game_mode</td>
<td>general.enter_game_mode</td>
</tr>
</tbody>
</table>
```

There are fewer errors, but some assets are still missing. The Asset Validation Gem (p. 371) seed-related commands can help find the missing assets.

4. Use the Asset Validation gem `addseedpath` command to add a likely missing bundle.

```
addseedpath levels\milestone2\level.pak
```

5. Enter the `listknownassets` command.
6. Examine the output. In the following example, the output shows missing button assets. In the case of the button assets, the bundle was packaged a while ago and must be repackaged. However, other assets are also still missing.

7. Add the missing assets to the seed list for the level.

8. Run the bundling commands (p. 394) for the level.

9. Drop the bundles into the Bundles directory.

10. Enter an assetbundlerbatch assetlists command, as shown in the following example. Use the --print argument to check the output. In the example, the single-line command has been formatted for readability.

```
assetbundlerbatch assetlists
   --addseed levels\milestone2\level.pak
   --addseed levels\milestone2\milestonecutscene.scriptevents
   --addseed levels\milestone2\hardcodedassetreference.luac
   --print
```

11. Verify that the output displays as expected.

12. Enter the assetbundlerbatch assetlists command again to bundle the assets, but this time without the --print argument. The example command is single-line, but has been formatted for readability.

```
assetbundlerbatch assetlists
   --addseed levels\milestone2\level.pak
   --addseed levels\milestone2\milestonecutscene.scriptevents
   --addseed levels\milestone2\hardcodedassetreference.luac
   --platform pc
   --assetlistfile DemoLevelList.assetlist
```

Output:

```
Saving Asset List file to ( G:\P4\dev\DemoLevelList_pc.assetlist )...
Save successful! ( G:\P4\dev\DemoLevelList_pc.assetlist )
```

13. Enter an assetbundlerbatch bundles command, as shown in the following example.
Tagging Files

```bash
assetbundlerbatch bundles
   --assetlistfile DemoLevelList_pc.assetlist
   --platform pc
   --outputbundlepath G:\P4\dev\DemoProject\Bundles\milestone2.pak
```

Output:

```
Creating Bundle ( G:\P4\dev\DemoProject\Bundles\milestone2_pc.pak )...
Bundle ( G:\P4\dev\DemoProject\Bundles\milestone2_pc.pak ) created successfully!
```

14. Enter the `loadbundles` command to reload the bundles, and then enter game mode.

```console
loadbundles
(CONSOLE) Executing console command 'loadbundles'
(BundlingSystem) - Loading bundles from bundles of type .pak
(BundlingSystem) - Opening bundle @assets@bundles\engine\dependencies.pak
(BundlingSystem) - Loaded bundle @assets@bundles\engine\dependencies.pak
(BundlingSystem) - Opening bundle @assets@bundles\milestone2_pc.pak
(BundlingSystem) - Loaded bundle @assets@bundles\milestone2_pc.pak
(BundlingSystem) - Opening bundle @assets@bundle\epasing.pak
(BundlingSystem) - Loaded bundle @assets@bundle\epasing.pak
general进入game mode
Returned;
[Warning] [Cloud Canvas] [Cloud Canvas] Resource manager was invoked but there are no deployments or resource groups.
Disable Accelerators
general.exit_game_mode
Returned;
general.exit_game_mode
Returned;
Enable Accelerators
```

All the assets loaded when game mode was entered are now in bundles.

### Using the File Tagging System to Include or Exclude Assets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard v1.22 and later use a file tagging system to include or exclude files at various stages of processing. This system employs file tag rules to select files that match specified patterns. File tags are associated with each rule to use as keys in file tag queries. This is done by using the Lumberyard FileTag API. You can create your own custom file tag rules using the Lumberyard Asset Editor. This is useful whenever you need additional control over which files should be included or excluded during a processing step. For example, as part of asset bundling, it is useful to eliminate “false positives” found after using the Missing Dependency Scanner.

**Topics**

- Creating File Tag Rules (p. 378)
- Using the FileTag API (p. 379)
Creating File Tag Rules

Use the Lumberyard Asset Editor to add custom file tag rules to either `exclude.filetag` or `include.filetag`, depending on whether you are excluding or including asset files. Both of these `.filetag` files are located in the `dev\Engine` directory. File tag rules consist of two required parts:

- **A File Pattern** that defines the files to match this rule. Supported patterns include:
  - **Exact** (for example, `readme.txt`)
  - **Wildcard** (for example, `*.cfxb`)
  - **Regex** (for example, `.*/gems?/?.*/gem.json`)
- One or more **File Tags** that are used by either Lumberyard or your own code as keys to refer to the file pattern matching defined by this rule.

You can use the **Comment** field to add more information about a file tag rule, for example to document its usage.

**Note**
Some file tags have designated uses within Lumberyard. Various tools may require that you use specific tags, such as `editoronly` and `shader`. You can find the full list of commonly used tags in the `FileTagsIndex` enum in `dev\Code\Framework\AzFramework\AzFramework\FileTag\FileTag.h`.

How to Create a File Tag Rule

1. In Lumberyard Editor, choose **Tools, Asset Editor**.
2. Choose **File, Open**, and select either `exclude.filetag` or `include.filetag` from the **Engine** directory.
3. Open **Definition**, find the line labeled **File Tag Map**, and click on the `'+' button to add a new child element.
4. Enter the desired file matching pattern in the **New Key** field.
5. Open your new file pattern key from the list and select the appropriate **File Pattern** type.
6. Add one or more **File Tags** to associate with your new file pattern.
7. Select **File, Save** to save your new file tag rule.

## Using the FileTag API

You can use the C++ FileTag API to write your own logic for determining whether to include or exclude files. The following example uses the file tagging system to ignore files that match patterns associated with the `ignore` and `shader` tags.

```cpp
bool IsIgnored(const char* szPath)
{
  using namespace AzFramework::FileTag;
  AZStd::vector<AZStd::string> tags{ "ignore", "shader" };

  bool shouldIgnore = false;
  QueryFileTagsEventBus::EventResult(shouldIgnore, FileTagType::exclude,
  &QueryFileTagsEventBus::Events::Match, szPath, tags);

  return shouldIgnore;
}
```

**Note**
In the previous example, it shows querying the `QueryFileTagsEventBus` on the ID `FileTagType::exclude`. This implies that this query is using the file tagging rules specified in the `exclude.filetag` file.
You can find the FileTag API in the \dev\Code\Framework\AzFramework\AzFramework\FileTag directory. In that directory, FileTag.h declares the Match method that was used in the previous example. There are other methods there, such as GetTags, which you can use to write more complex logic. You may find it useful to work with the excludeFileComponent helper class, found in FileTagComponent.h. This component class automatically loads the default exclusion file for you, sets the file tag type to FileTagType::exclude, and connects to the QueryFileTagsEventBus upon activation.

Amazon Lumberyard Asset List Comparison Operations

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Asset list comparisons are rules provided to the AssetBundlerBatch.exe tool to determine which files should be included or excluded from the final bundle asset list. The asset list files have the suffix .assetlist and contain a flat list of paths and names of asset files. The following operations are available to use from the Asset Bundler tool:

- Delta Comparison Operation (p. 380)
- Union Comparison Operation (p. 381)
- Intersection Comparison Operation (p. 381)
- Intersection Count Comparison Operation (p. 382)
- Complement Comparison Operation (p. 383)
- File Pattern Comparison Operation (p. 384)

Note:

While the comparison operations use terms from set theory, they are not exactly the same as the actual set operations. This is particularly true of the delta comparison operation (p. 380), which includes a file entry in both sets if one of those files has been modified in the second set. Additionally, file pattern matching is not a set operation.

Asset List Delta Comparison Operation

This operation takes two asset list files to create an asset list for a bundle with only the assets that you need to ship in your release bundle. Use this operation to create bundles for incremental updates, such as delta patches for levels. To use this operation, run AssetBundlerBatch.exe with a --comparisonType argument value of 0 or delta.

To perform a delta comparison operation, open a command prompt, and run the following command:

```
 AssetBundlerBatch.exe compare ^
 --comparisonType delta ^
 --firstAssetFile firstAssetList_pc.assetlist ^
 --secondAssetFile secondAssetList_pc.assetlist ^
 --output deltaAssetList.assetlist
```

The following diagram shows the delta comparison operation.
Comparison Operations

Asset List Union Comparison Operation

This operation takes two asset list files to create an asset list for a bundle that combines all the assets from both lists. It includes only the modified version of a file in the output asset list, not the original from the first asset list. Use this operation when you have two bundles that no longer need to be separate and should be combined into a single bundle. To use this operation, run `AssetBundlerBatch.exe` with a `--comparisonType` argument value of 1 or `union`.

To perform a union comparison operation, open a command prompt, and run the following command:

```
AssetBundlerBatch.exe compare ^
   --comparisonType union ^
   --firstAssetFile firstAssetList_pc.assetlist ^
   --secondAssetFile secondAssetList_pc.assetlist ^
   --output unionAssetList.assetlist
```

The following diagram shows the union comparison operation.

Asset List Intersection Comparison Operation

This operation takes two asset list files to create a bundle asset list with only items that are in both asset lists. To use this operation, run `AssetBundlerBatch.exe` with a `--comparisonType` argument value of 2 or `intersection`.

Version 1.28
To perform an intersection comparison operation, open a command prompt and run the following command:

```
AssetBundlerBatch.exe compare ^
--comparisonType intersection ^
--firstAssetFile firstAssetList_pc.assetlist ^
--secondAssetFile secondAssetList_pc.assetlist ^
--output intersectionAssetList.assetlist
```

The following diagram shows the intersection comparison operation.

![Intersection Comparison Operation Diagram]

**Asset List Intersection Count Comparison Operation**

This operation takes any number of asset files to create a bundle asset list with only items that appear a given number of times across all of the asset lists. This comparison type can't be used as part of a series of comparison rules, and requires the use of the `--intersectionCount` argument. To use this operation, run `AssetBundlerBatch.exe` with the `--comparisonType` value of 5 or `intersectionCount`.

To perform an intersection count comparison operation, open a command prompt and run the following command:

```
AssetBundlerBatch.exe intersectionCount ^
--comparisonType intersectionCount ^
--intersectionCount 3 ^
--firstAssetFile
  firstAssetList_pc.assetlist, secondAssetList_pc.assetlist, thirdAssetList_pc.assetlist ^
--output intersectionCountAssetList.assetlist
```

The following diagram shows the intersection comparison operation.
Comparison Operations

Asset List Complement Comparison Operation

This operation takes two asset list files to create a bundle asset list with each item in the second asset list that is not in the first list. It works like the delta comparison, except that it does not check the file hash and will not include modified versions of a file that is in both lists. To use this operation, run `AssetBundlerBatch.exe` with a `--comparisonType` argument value of 3 or `complement`.

To perform a complement comparison operation, open a command prompt, and run the following command:

```
AssetBundlerBatch.exe compare ^
--comparisonType complement ^
--firstAssetFile firstAssetList_pc.assetlist ^
--secondAssetFile secondAssetList_pc.assetlist ^
--output complementAssetList.assetlist
```

The following diagram shows the intersection comparison operation.

Intersection Count Comparison Operation

Asset List A

- FileA.txt
- FileB.txt

Asset List B

- FileA.txt
- FileB.txt
- FileC.txt

Result Asset List

- FileB.txt

count=3

Asset List C

- FileB.txt
- FileC.txt

Asset List Complement Comparison Operation

This operation takes two asset list files to create a bundle asset list with each item in the second asset list that is not in the first list. It works like the delta comparison, except that it does not check the file hash and will not include modified versions of a file that is in both lists. To use this operation, run `AssetBundlerBatch.exe` with a `--comparisonType` argument value of 3 or `complement`.

To perform a complement comparison operation, open a command prompt, and run the following command:

```
AssetBundlerBatch.exe compare ^
--comparisonType complement ^
--firstAssetFile firstAssetList_pc.assetlist ^
--secondAssetFile secondAssetList_pc.assetlist ^
--output complementAssetList.assetlist
```

The following diagram shows the intersection comparison operation.

Complement Comparison Operation

Asset List A

- FileA.txt
- FileB.txt

Asset List B

- FileA.txt
- FileB.txt
- FileC.txt
- FileC.txt

Result Asset List

- FileC.txt
Asset List File Pattern Operation

This operation takes an asset list file, and a file pattern to apply. Any files that match this pattern in the asset list will be included in the output asset list. To use this operation, run AssetBundlerBatch.exe with a --comparisonType argument value of 4 or filepattern.

To perform a file pattern comparison operation, open a command prompt and run the following command:

```
AssetBundlerBatch.exe compare ^
--comparisonType filepattern ^
--filePatternType 0 ^
--filePattern "*.xml" ^
--firstAssetFile assetList_pc.assetlist ^
--output filePatternAssetList.assetlist
```

Note:

The previous command looks for files that have the .xml suffix for inclusion. You can replace it with any wildcard- or regex-based file pattern that you want to use for comparison.

The following diagram shows the file pattern comparison operation.

File Pattern Operation

How to Perform Multiple Asset List Comparison Operations

The following guidance shows the process of creating an asset list for a game patch that only contains modified and updated text (.txt) file content. During the process, it also validates the asset list contents against an inclusion list generated as the result of an intersection comparison.

The following diagram shows the comparison process and the outputs for this example.
Prerequisites

To complete the procedures in this tutorial, make sure that you have the following set up:

- An installed and configured installation of Amazon Lumberyard v1.22 or later. Download the latest version of Amazon Lumberyard.
- A Lumberyard game project ready to build and compile. You can use the Starter Game sample if you don't have a game project. Learn more about the Starter Game sample project. (p. 148).

Setup

Create the files that you will use in this tutorial.

1. In your game project's root folder, create these empty files:
   - FileA.txt
   - FileB.txt
   - FileC.txt
   - FileD.txt
   - FileE.cfg
   - FileF.cfg
   - do-not-add-me.txt
2. Process the assets on your project by running the Asset Processor (p. 248).

Create asset list files for the comparison operation

Create the .seed and .assetlist files to use in your comparisons.

1. Open a command prompt, navigate to the directory where you output the asset list files, and run the following command to create your initial seed list:

   AssetBundlerBatch.exe seeds ^
Comparison Operations

2. Run this command to create your asset list from the seed list:

```
AssetBundlerBatch.exe assetlists ^
--seedListFile include-list.seed ^
--assetListFile include-list.assetlist
```

3. Run this command to create your v1 seed file:

```
AssetBundlerBatch.exe seeds ^
--addSeed FileA.txt,FileB.txt ^
--seedListFile mygame_v1.seed
```

4. Run this command to create your v1 asset list file:

```
AssetBundlerBatch.exe assetlists ^
--seedListFile mygame_v1.seed ^
--assetListFile mygame_v1.assetlist
```

5. Run this command to create your v2 seed file:

```
AssetBundlerBatch.exe seeds ^
--addSeed FileA.txt,FileB.txt,FileC.txt,fileE.cfg,fileF.cfg,do-not-add-me.txt ^
--seedListFile mygame_v2.seed
```

6. Run this command to create your v2 asset list file:

```
AssetBundlerBatch.exe assetlists ^
--seedListFile mygame_v2.seed ^
--assetListFile mygame_v2.assetlist --print
```

This last command should produce output that looks like this, enabled by the `--print` flag:

```
Printing assets for Platform ( pc ):  
- filea.txt 
- fileb.txt 
- filec.txt 
- filee.cfg 
- filef.cfg 
- do-not-add-me.txt 
Total number of assets for Platform ( pc ): 6.
```

You have now created the `.seed` and `.assetlist` files that you need to start the comparison operations and assemble a final bundle asset list.

**Note**

At this point, you can choose either one of the next two steps to complete the tutorial. To run each comparison command individually, see the steps in section 1.6.4. To run all of the comparison operations in one command, see the steps in section 1.6.5.

**Run the individual step comparison commands**

1. Open a command prompt and navigate to the directory where you output the asset list files. Run the following command:
AssetBundlerBatch.exe compare ^
--comparisonType delta ^
--firstAssetFile mygame_v1_pc.assetlist ^
--secondAssetFile mygame_v2_pc.assetlist ^
--output multistep_delta.assetlist ^
--print

In this step the files that are in the v1 asset list, fileA.txt and fileB.txt, are removed based on comparisonType 0, the delta comparison type. Your command output should look like this:

Printing assets from the comparison result {PATH TO YOUR LUMBERYARD ROOT HERE}\dev \multistep_delta_pc.assetlist.
---------------------------------------------------------------
- filee.cfg
- do-not-add-me.txt
- filec.txt
- filef.cfg
Total number of assets (4).
---------------------------------------------------------------
Saving results of comparison operation...
Save successful!

2. Remove any files from the asset list that aren’t in the inclusion list by running this command from the prompt:

AssetBundlerBatch.exe compare ^
--comparisonType intersection ^
--firstAssetFile include_pc.assetlist ^
--secondAssetFile multistep_delta_pc.assetlist ^
--output multistep_include.assetlist ^
--print

For this step, the file do-not-add-me.txt is removed because it isn’t in the generated inclusion list, based on specified the intersection comparison (comparisonType 2). Your command output should look like this:

Printing assets from the comparison result {PATH TO YOUR LUMBERYARD ROOT HERE}\dev \multistep_include_pc.assetlist.
---------------------------------------------------------------
- filec.txt
- filee.cfg
- filef.cfg
Total number of assets (3).
---------------------------------------------------------------
Saving results of comparison operation...
Save successful!

3. Run this command to remove anything that is not a text (.txt) file from the asset list:

AssetBundlerBatch.exe compare ^
--comparisonType filepattern ^
--filePatternType 0 ^
--filePattern *.txt ^
--firstAssetFile multistep_include_pc.assetlist ^
--output multistep_filepattern.assetlist ^
--print
Comparison Operations

For this step, the two .cfg files are removed because they don't match the file pattern, based on the file pattern comparison (comparisonType 4). Your command output should look like this:

```
Printing assets from the comparison result \{PATH TO YOUR LUMBERYARD ROOT HERE}\dev \multistep_filepattern_pc.assetlist.
- filec.txt
  Total number of assets (1).
Saving results of comparison operation...
Save successful!
```

Run the multiple step comparison command

You can also perform all of these commands in a single command. The multiple step command is equivalent to running all of the commands in the prior steps individually, and uses variables you specify to store the intermediate results of the comparisons.

In this example, the multiple step comparison command runs the 3 comparisons sequentially, storing the results in variables that are used in the next steps. The delta comparison between v1 and v2 is stored into the $delta_all variable, and then runs the intersection comparison on the inclusion list with the asset list $delta_all, storing the results in the $delta_include variable. Finally, the file pattern comparison is run against all text files stored in the $delta_include variable. The output of this final command is stored in the file, mygame_v1tov2_patch.assetlist.

To try this approach, open a command prompt, and navigate to the directory where you output the asset list files. Run the following command:

```
AssetBundlerBatch.exe compare ^
  --comparisonType delta,intersection,filepattern ^
  --filePatternType 0 ^
  --filePattern *.txt ^
  --firstAssetFile mygame_v1_pc.assetlist,include_pc.assetlist,$delta_include ^
  --secondAssetFile mygame_v2_pc.assetlist,$delta_all ^
  --output $delta_all,$delta_include,mygame_v1tov2_patch.assetlist ^
  --print
```

The command should produce output that looks like this:

```
Printing assets from the comparison result \{PATH TO YOUR LUMBERYARD ROOT HERE}\dev \mygame_v1tov2_patch_pc.assetlist.
- filec.txt
  Total number of assets (1).
Saving results of comparison operation...
Save successful!
```

You can also confirm that the operation was successful by opening mygame_v1tov2_patch.assetlist in a text editor and checking that it only contains the files you expect to see.

How multiple comparisons work

When running multiple step commands, use a comma-separated list for each relevant parameter. The individual steps in the command match up to their placement in this comma-separated parameter value list.
Default dependencies

The previous example uses three comparison operations. The first two comparisons reference a first and second asset list file, and the last comparison references a file pattern and the first asset file. The multiple step command broken down into its component parts looks like this:

**Breaking down the process of a single multiple comparison command**

<table>
<thead>
<tr>
<th>Command Parameter</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparisonType</td>
<td>delta</td>
<td>intersection</td>
<td>filepattern</td>
</tr>
<tr>
<td>firstAssetFile</td>
<td>mygame_v1_pc.assetlist incluedelta_include (temp file)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>secondAssetFile</td>
<td>mygame_v2_pc.assetlist</td>
<td></td>
<td>$delta_all</td>
</tr>
<tr>
<td>filePatternType</td>
<td>N/A</td>
<td>N/A</td>
<td>Wildcard (parameter value 0)</td>
</tr>
<tr>
<td>filePattern</td>
<td>N/A</td>
<td>N/A</td>
<td>*.txt</td>
</tr>
<tr>
<td>output</td>
<td>$delta_all</td>
<td>$delta_include</td>
<td>mygame_v1tov2_patch.assetlist</td>
</tr>
</tbody>
</table>

Read this table horizontally to see the data supplied to each comparison during each step of the process. Read this table vertically to see the parameter used for each command for each step.

**Default dependencies for Lumberyard projects**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com) or visit the [AWS Game Tech blog](https://aws.amazon.com) to learn more.

Throughout your Lumberyard project you'll use assets across multiple levels, or find those that need to be included whether or not they're a strict dependency. To handle these use cases, Lumberyard supports **default dependency files** which define assets that are always required when bundling your project. Default dependencies are also used by Gems to make sure their own critical assets are always included.

Default dependencies give you a convenient way to list assets that should be bundled as part of your whole project, and can be applied when generating any asset list by using `--addDefaultSeedListFiles`. When you use this argument as part of an asset bundler command, it picks up the default dependencies for the Lumberyard engine, included gems, and your project.

**Default dependency file locations**

There are four levels of default dependency files available. All paths are relative to `lumberyard_install\dev`:

<table>
<thead>
<tr>
<th>Default dependency</th>
<th>File path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td>Engine\Engine_Dependencies.xml</td>
<td>The dependencies packaged for every Lumberyard project. Only edit this file when you use the same installation to create multiple projects that need to include a specific resource which even basic game functionality depends on.</td>
</tr>
</tbody>
</table>
Default dependencies

<table>
<thead>
<tr>
<th>Default depende</th>
<th>File path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gems</td>
<td>Gems\gem_name\Assets \gem_name_Dependencies.xml</td>
<td>The required dependencies for the named gem. When creating a new gem, include any resources that are required regardless of whether they’re used explicitly within a project here. Never edit the default dependency file for a gem which you aren’t writing or customizing.</td>
</tr>
<tr>
<td>Project</td>
<td>project_name\project_name_Dependencies.xml</td>
<td>Project-wide dependencies. This is the default dependency file that you'll be editing most frequently, and should include things like game-wide audio, configuration information for pre-loading resources at launch time, or other assets that must always be included with your project. When you create a new project, the dependency file is created from the ProjectTemplates \DefaultTemplate${ProjectName}${ProjectName}_Dependencies.xml template.</td>
</tr>
</tbody>
</table>

**Important**

These files are not seed lists, and can’t be manipulated with commands that modify seed lists. Default dependencies are built by the Asset Processor to create seed lists in the cache. These seed lists are then picked up by the Asset Bundler when you use the --addDefaultSeedListFiles flag.

Another important consequence of this is that every time you change a default dependencies file, it must be built by the Asset Processor to generate an updated seed list for the Asset Bundler.

Default dependencies file format

Default dependencies files are written in XML and consist of only two elements: EngineDependencies and Dependency. The root element of a default dependency file should always be <EngineDependencies version="1.0.0">, including this specific value for the version attribute.

Dependency elements are the only children of the EngineDependencies node, and require two attributes: path and optional. Dependency elements have no children, including no text content.

How each Dependency node is treated depends on the attribute values:

- optional – This value is either true or false, and describes whether or not the listed asset is absolutely required by the asset bundling process. Most often you’ll want to use true for asset paths without wildcards, and false for paths which contain a wildcard.
- path – This value is the path to the asset(s) to include, and accepts the use of the * wildcard character. Wildcards search all subdirectories recursively.

Because wildcard matching may catch files that you don’t want to include as default assets, you can add exclusion paths: Dependencies where the path starts with a : character. Assets in an exclusion path are not included as a default dependency.

The following is an example of a default dependency list for a project, commented to make it clear which dependencies are processed by each node:

```xml
<EngineDependencies versionnumber="1.0.0">
  <Dependency path="libs/particles/preloadlibs.txt" optional="true" /> <!-- Include particle pre-loads if present -->
</EngineDependencies>
```
Asset Bundler Concepts and Terms

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following terms and concepts relate to the Lumberyard Asset Bundler:

Asset ID

All product assets are assigned an Asset ID. The asset ID consists of the source asset’s UUID plus a sub ID, which uniquely identifies a product. Sub IDs are usually generated by an asset builder. They must be stable for running in a build, and mutually exclusive. Each product should have a unique sub ID that does not change. The Asset ID data structure is defined in dev\Code\Framework\AzCore\AzCore\Asset\AssetCommon.h. See also Source GUID.

Asset Browser

The Asset Browser program, which is part of the Lumberyard Editor, shows the current state of your assets. You can use this program to navigate through a set of assets, such as tracking which assets have been processed and which will be processed.

Asset Bundle

An asset bundle consists of product assets grouped together in a single compressed file. The bundle is mounted in a Lumberyard game so that the assets can be loaded and used during runtime. Asset bundles are .pak files, which use the CryEngine .pak file format. This file format is similar to the .zip file format.

For more information, see Creating a Custom Asset Builder (p. 318).

Asset List

An asset list contains information for product assets. This file is generated using the Asset Bundler. The information includes a fingerprint for each asset in the list. The fingerprint includes a modification time and file hash. An asset list can be generated from seed lists, individual seeds, or by using comparison operations on other asset lists. Asset list files have the file extension .assetlist. See also Seed and Seed List.

Builder

An asset builder is a C++ module that is used to transform source assets into product assets. For example, a builder that handles textures might also transform a source asset PNG file to a product asset DDS file. Builders emit metadata about source and product assets, such as dependencies.

For more information, see Creating a Custom Asset Builder (p. 318).

Copy Job

A copy job is an asset processing task that copies a source asset to the cache without modifying it. There are two ways that copy jobs occur:

• When you are authoring a builder. If the builder emits a source asset as a product asset, the Asset Processor copies the source to the cache.
When using additional asset processing rules as defined in AssetProcessorPlatformConfig.ini. These additional rules can include copy jobs, and are used to copy many file types to the cache.

As a best practice, create an Asset Builder for your assets so you can add product dependencies. While it’s easy to add copy jobs to AssetProcessorPlatformConfig.ini, it can be easy to forget to update files for dependencies later.

### Leaf Asset

A "leaf" asset is a product asset that has no dependencies on other product assets. See also Product Dependency.

### Product Asset

A product asset is game content that is loaded and used in the runtime game. A product asset is generated from a source asset during asset processing using a builder. The product asset is often a modified version of the source asset, such as image content that is compressed and otherwise optimized for runtime. Product assets reference their product dependencies. See also Source Asset.

When running a game using the Lumberyard asset system, only product assets are available. The source assets are not available. This includes running the game in the Lumberyard Editor or in a platform-specific launcher. When packaging a game using the Asset Bundler, only product assets are bundled. Product assets are available only from the asset cache or from asset bundles.

**Note**

When working with slices, keep in mind that only dynamic slices can be used at run-time. Non-dynamic slices, even if they are product assets, should not be included in bundles.

For more information about the asset life cycle, see Working with the Asset Pipeline and asset files (p. 247).

### Product Dependency

A product dependency occurs when one product asset references another product asset. When bundling assets, both the referencing asset and the referenced asset should be included in the bundle. Dependencies are created by referencing the Asset ID of the product asset. See also Source Dependency.

Product dependencies are generally relevant during asset packaging and bundling. With product dependencies established, bundling becomes the task of "shipping game level 1 and everything it references". For example, you have a character that is spawned through a dynamic slice. This character has a model, and the model has a material, which has a texture. In this scenario, the material needs the texture loaded to work, so the material has a product dependency on the texture. The model needs the material loaded to work, so the model has a product dependency on the material. The character needs the model loaded to work, so the dynamic slice for the character has a product dependency on the model. When bundling, you can include the character and expect that all dependencies are included.

### Root Folder

The root folder concept is inherited from CryEngine. When a Lumberyard application starts, it loads and initializes a set of modules, or code libraries, to support the application’s gems. Legacy modules, such as bootstrap.cfg and engine.cfg, are loaded from the root folder. Newer AZ modules are loaded from the assets folder.

For more information, see Programming with Gems (p. 1073).

### Runtime

A standalone game is referred to as the runtime. It can be run through a platform-specific launcher, such as a console, or in the Lumberyard Editor. A runtime uses the Lumberyard asset system to access only those product assets that were bundled with the game.
Scan Folder

Scan folders are scanned for assets during game runtime. A set of scan folders are designated in the AssetProcessorPlatformConfig.ini file. Scan folders are defined when the Asset Processor starts. They cannot be added, removed, or renamed during runtime.

Seed

A seed is a product asset added to a seed list. Defining a product asset as a seed can make bundling assets or mounting into a game runtime more efficient. Seeds can also be useful when validating or debugging an asset bundle, such as using "seed mode" in the Asset Validation Gem to track asset dependencies. See also Seed List.

Seed List

A seed list is a .seed file containing a collection of seeds for the bundling system to place into an asset list. Seed lists are also used to build asset dependency charts. See also Asset List File.

For simple game packages, seed lists are commonly used with level files. For example, you have a game with three levels. Levels 1 and 2 are production levels, and the other is the development test level. You want to ship your game with the two production levels and all files referenced by them. You can use the Asset Bundler to create a MyGame.seed asset list file, and then add your Level 1 and Level 2 product assets as seeds to this file.

Source Asset

A source asset is an instance of an asset that is not associated with a specific product. When working with an asset, you edit the source asset and then use an asset builder to generate a product-specific instance of the asset. When authoring edit-time code, you may need to access editor-only product assets or source assets. If your logic is actually editing a file, use the source asset. If your logic is only reading the file, use the editor-only product asset in the asset cache. Source assets are stored in the source asset folder, usually under dev\ProjectName\.

Source Dependency

A source dependency occurs when one source asset references another source asset. Source assets list each dependency by referencing the ID of the source asset. See also Product Dependency.

In general, source dependencies are relevant during content creation at edit time and when developing tools pipelines. Source dependencies are used in the following scenarios:

- When an asset is changed, all assets that depend on the changed asset must be updated.

  An asset won't process until all of the source assets that it depends on finish processing.

  For example, you're working with models and materials. Model source assets depend on material source assets. However, the product asset for the model does not need a corresponding product asset for the material. In this scenario, the relevant material must be generated before processing the model source asset, which is an FBX file, to output the product asset as a CGF file.

Source GUID

A source GUID is used to identify assets. It is generated by hashing the path—relative to the scan folder—of the source asset file. It is technically a UUID value, but is commonly referred to as a GUID. See also Asset ID.

Source assets are associated with a gem, and source GUIDs are unique within a gem's set of assets. It is possible for source assets from different gems to have the same GUID, if they have the same relative path, file name, and extension. When referencing a source GUID, the file from the highest priority scan folder is returned for all queries and copied to the cache.

Relative Paths

Some of the Lumberyard tool operations take relative paths as inputs. There are two types of relative paths:
• **Cache-relative paths** – Tells various systems where to find pre-processed assets in your asset cache. Each Lumberyard tool that uses cache-relative paths will be relative to a different subdirectory of the asset cache. For example, a cache-relative path from the absolute path `C:\Lumberyard\dev\Cache\SamplesProject\pc\samplesproject\levels\samples\advanced_rinlocomotion\level.pak` that does not include the platform and project would be `levels\samples\advanced_rinlocomotion\level.pak`.

• **Engine-root-relative paths** – Specifies files located in directories under `C:\Lumberyard\dev\`. For example, an engine-root-relative path lets you represent the location `C:\Lumberyard\dev\SamplesProject\textures\UIEditor_Sample\ButtonNormal.tif` as `SamplesProject\textures\UIEditor_Sample\ButtonNormal.tif`.

For more information, see Asset IDs and File Paths (p. 325).

---

**Lumberyard Asset Bundler Command-Line Tool Reference**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Contents**

- **General Use** (p. 395)
  - Options (p. 395)
- **Seed lists - seeds** (p. 396)
  - Options (p. 396)
  - Examples (p. 397)
- **Asset Lists - assetLists** (p. 398)
  - Options (p. 398)
  - Examples (p. 400)
- **Comparison rules - comparisonRules** (p. 401)
  - Options (p. 401)
  - Examples (p. 402)
- **Comparisons - compare** (p. 402)
  - Options (p. 402)
  - Examples (p. 404)
- **Bundle settings - bundleSettings** (p. 405)
  - Options (p. 405)
  - Examples (p. 406)
- **Asset bundles - bundles** (p. 406)
  - Options (p. 406)
  - Examples (p. 407)
- **Bundle from seed - bundleSeed** (p. 407)
  - Options (p. 407)
  - Examples (p. 408)

The Amazon Lumberyard Asset Bundler is driven by a command-line tool called `AssetBundlerBatch`. This tool manages seed lists, asset lists, comparisons, and asset bundles. The asset bundler is **not used**
to compile assets into the format used by the bundler for distribution — that is the role of the Asset Processor. Before running the Asset Bundler, make sure that you:

- Enable each platform assets should be bundled for. Enabled platforms are managed by editing the `dev\AssetProcessorPlatformConfig.ini` file in your project.
- Run the Asset Processor to ensure that assets and their metadata are up to date.

**Note**

All example commands in this reference are executed from the `dev\` folder of a project and assume a build produced with Visual Studio 2017 on Windows. The executable path and path separators will be different on other platforms.

See the [Asset Bundler Concepts and Terms](p. 391) or [Glossary](p. 3281) for definitions of terms used in this reference.

## General Use

The format for `AssetBundlerBatch` commands is:

```
AssetBundlerBatch command --parameterWithArgs arg1,arg2 --flagParameter ... 
```

The `AssetBundlerBatch` executable is contained in the `dev\Bin64\HostPlatform` folder of your project. For example, when building on PC with Visual Studio 2017, the asset processor is located at `dev\Bin64\vc141`.

The elements in this example invocation break down to:

- **command** - The command for the asset bundler to run. Examples include `seeds` and `assetLists`.
- **--parameterWithArgs** - An argument which takes parameters. If a parameter can take more than one argument, you can either separate arguments with a `,` character without using whitespace, or by giving the parameter multiple times:
  - `--parameterWithArgs arg1,arg2`
  - `--parameterWithArgs="arg1,arg2"`
  - `--parameterWithArgs arg1 --parameterWithArgs arg2`
  - `--parameterWithArgs="arg1" --parameterWithArgs="arg2"

  These styles of writing parameters can be freely mixed and matched. Note that not all arguments take more than one input parameter.
- **--flagParameter** - A flag which doesn't take any arguments. Flags represent boolean values and turn features of the Asset Bundler on and off.

An example command is:

```
Bin64\vc141\AssetBundlerBatch seeds --seedFileList MyProject\AssetBundling\SeedLists\AllDependencies.seed ^
--addPlatformToSeeds ^
--platform ios,pc
```

## Options

The options in this section are valid for all Asset Bundler subcommands.
--help, -h

Without a command, prints out the help text for AssetBundlerBatch. When used with a command, prints the help information for that command.

_Type:_ Flag

_Required:_ No

--verbose

Enables more detailed output messages. _Error_ and _Warning_ messages will display the file name and line number which generated the message. This flag is intended for use when looking at source files or generating output for debugging purposes.

_Type:_ Flag

_Required:_ No

Seed lists - seeds

The _seeds_ command is used to manage seed lists, the first phase of the Asset Bundler workflow. Seed list files have the `.seed` extension.

This command requires an existing cache of assets for each provided platform. To make sure that the cache is up to date, update the supported platforms and run the Asset Processor.

Options

--seedListFile

The seed list to modify and save. This file must be writable and have the `.seed` extension. The file is created if it doesn't exist. The argument's value may be either an absolute or engine-root-relative path.

_Type:_ Single-value argument

_Required:_ Yes

--addSeed

Add seeds to the seed list for the _--platform_ values. If a seed does not exist for one of the specified platforms, then the invalid platforms are ignored. The argument's value may be any number of cache-relative paths to pre-processed assets.

_Important_

Although `.slice` files are built and present in the asset cache, they aren't product assets and don't generate dependencies when added to a seed list. Avoid using them as seeds.

_Type:_ Multi-value argument

_Required:_ No

--removeSeed

Remove the _--platform_ values for the provided seeds. If a seed is not available for a platform, it's ignored. A seed is not completely removed from the seed list until it has no associated platforms.

_Type:_ Multi-value argument

_Required:_ No
--addPlatformToSeeds

Adds the --platform values to all seeds in the seed list. If a seed isn't valid for a provided platform, a warning is printed indicating the affected seed and the platform which caused the warning. The seed is updated for all other valid platforms.

Type: Flag
Required: No

--removePlatformFromSeeds

Removes the --platform values from all seeds in the seed list. If this would remove all platforms for an existing seed, the seed is unchanged and a warning is printed. Use --removeSeed to remove a seed from a seed list.

Type: Flag
Required: No

--print

Prints the contents of the seed list after performing operations.

Type: Flag
Required: No

--platform

The platforms used for this command. Defaults to all supported platforms for the current project. Supported platforms can be changed by modifying AssetProcessorPlatformConfig.ini.

Platform names can be found in the AssetProcessorPlatformConfig.ini file, or as folder names found under dev\Cache\ProjectName.

Type: Multi-value argument
Required: No

--updateSeedPath

Updates the relative paths for every seed in the seed list.

Type: Flag
Required: No

--removeSeedPath

Removes the relative path hints for every seed in the seed list. This argument is useful when you want to share your seed list with a third party.

Type: Flag
Required: No

Examples

In the following examples, assume that the current project has enabled the platforms pc, ios, and es3.

Example Add seed

Create the seed list testFile.seed if it doesn't exist, and adds the asset1.pak and asset2.pak assets as seeds for the PC platform:
Example Add platforms

Add the ios and es3 platforms for all seeds in the testFile.seed seed list:

```
Bin64vc141\AssetBundlerBatch.exe seeds --seedListFile testFile.seed --addPlatformToSeeds --platform ios,es3
```

Example Display seed list contents

Show the contents of the testFile.seed file, including the absolute and relative paths of all assets used as seeds:

```
Bin64vc141\AssetBundlerBatch.exe seeds --seedListFile testFile.seed --print
```

Asset Lists - assetLists

The assetLists command is used to manage and create asset lists. See the --assetListFile argument for how platform information is encoded in asset list file names. Asset list files have the .assetlist extension.

This command requires an existing cache of assets for each provided platform. To make sure that the cache is up to date, update the supported platforms and run the Asset Processor.

Options

--assetListFile

The base name of the platform-specific asset list files to be generated. This file's path must be writable, and the file name must have the .assetlist extension. The argument's value may be either an absolute or engine-root-relative path.

The asset list files generated are named based on the value of this argument and the provided platforms. For an argument value of path\assetFile.assetlist, each platform gets an asset file created as path\assetFile_Platform.assetlist. For example, assetFile.assetlist for the platform pc is named assetFile_pc.assetlist.

assetLists invocations must contain either the --assetListFile argument or --print flag.

_Type: Single-value argument

_Required: No_

--seedListFile

The seed list used to generate the asset list. This argument can be used along with other arguments that provide seeds.

_Type: Single-value argument

_Required: No_

--addSeed

Individual seeds used to generate the asset list. This argument can be used along with other arguments that provide seeds.

_Type: Single-value argument

_Required: No_
Important
Although \textit{.slice} files are built and present in the asset cache, they aren't product assets and don't generate dependencies when used as seeds. Avoid using them as seeds.

\textit{Type}: Multi-value argument

\textit{Required}: No

\texttt{--addDefaultSeedListFiles}

Automatically include the default seed lists for the Lumberyard Engine, default project, and all enabled Gems. As paths relative to \texttt{lumberyard\_dev}, the files that define default assets are:

- \textbf{Engine} – \texttt{Engine\Engine\_Dependencies\.xml}
- \textbf{Gems} – \texttt{Gems\gem\_name\Assets\gem\_name\_Dependencies\.xml}
- \textbf{Project} – \texttt{project\_name\project\_name\_Dependencies\.xml}

By default, the project dependencies includes pre-loaded particle libraries and game-wide audio, excluding level-specific audio.

This argument can be used along with other arguments that provide seeds.

\textit{Type}: Flag

\textit{Required}: No

\texttt{--skip}

A list of assets to ignore. This can include both seed assets and any dependencies that were picked up by the asset processor. The argument value is a comma-separated list of cache-relative paths to assets that have already been pre-processed.

\textit{Type}: Multi-value argument

\textit{Required}: No

\texttt{--platform}

The platforms used for this command. Defaults to all supported platforms for the current project. Supported platforms can be changed by modifying \texttt{AssetProcessorPlatformConfig.ini}.

Enabled platform names can be found in the \texttt{AssetProcessorPlatformConfig.ini} file, or as directories under \texttt{dev\Cache\ProjectName}. 

\textit{Type}: Multi-value argument

\textit{Required}: No

\texttt{--print}

Output a list of all product dependencies. The behavior of this argument depends on which other arguments are provided to the \texttt{AssetBundlerBatch argumentList} command:

- \texttt{--assetListFile} without operations: Read an existing asset list from disk and display its contents.
- \texttt{--assetListFile} with operations: Generate a new asset list and display its contents.
- Without \texttt{--assetListFile}: Display the contents of the asset list that would be generated.

Each of these behaviors is illustrated in the \texttt{assetList examples} (p. 400).

\texttt{AssetBundlerBatch assetLists} commands must contain either the \texttt{--assetListFile} argument to generate new asset lists, or the \texttt{--print} flag to write information to the console.

\textit{Type}: Flag
Required: No
--dryRun

Run without generating new asset lists.

Type: Flag

Required: No
--generateDebugFile

Generate a file that contains additional information about asset inclusion for debugging purposes. This file contains a hierarchical list of every asset contained within the asset list file, as well as information about which seeds marked each specific asset as a product dependency. The debug file is stored in the same path as the --assetListFile, with the extension .assetlistdebug.

This argument requires the use of --assetListFile. If you use the --print argument, the output of the generated files is displayed in the console, not the output of the debug file.

Type: Flag

Required: No
--allowOverwrites

Allow overwriting of existing asset files. By default, existing asset lists will not be regenerated.

Type: Flag

Required: No

Examples

In the following examples, assume that the seed list testFile.seed exists and that the pc, ios, and es3 platforms are enabled.

Example Display default assets

Display which asset lists would be generated from the seed lists for the Lumberyard Engine and enabled Gems for a project's default platforms:

Bin64vc141\AssetBundlerBatch.exe assetLists --addDefaultSeedListFiles --print

Example Display asset lists for default platforms

Use an input asset list and the project's default platforms to display that asset list's contents:

Bin64vc141\AssetBundlerBatch.exe assetLists --assetListFile assetListFile.assetlist --print

Example Create an asset list and debug file from a seed list

Generate an asset list testList_pc.assetlist and debug information testList_pc.assetlistdebug from the testFile.seed seed list:

Bin64vc141\AssetBundlerBatch.exe assetLists --assetListFile testList.assetlist ^
--seedListFile testFile.seed ^
--platform pc ^
--generateDebugFile
Example Display asset list contents for a platform

Display the contents of the testList_pc.assetlist file:

```bash
Bin64vc141\AssetBundlerBatch.exe assetLists --assetListFile testList.assetlist --platform pc --print
```

Example Regenerate asset lists from a seed list

Regenerate all asset lists from the testFile.seed seed list, overwriting the testList_pc.assetlist file if it exists:

```bash
Bin64vc141\AssetBundlerBatch.exe assetLists --assetListFile testList.assetlist --seedListFile testFile.seed --allowOverwrites
```

Comparison rules - comparisonRules

The comparisonRules command is used to generate comparison rules files. Comparison rules files are used as inputs for the compare (p. 402) subcommand. Comparison rules files are pre-built descriptions of which operations to perform and in what order. For more information on comparison rules, see Amazon Lumberyard Asset List Comparison Operations (p. 380).

Options

--comparisonRulesFile

The comparison rules file to generate.

Type: Single-value argument

Required: Yes

--comparisonType

The comparison types to apply, in the given order. Valid values are:

- 0 or delta: Delta comparison
- 1 or union: Union
- 2 or intersection: Intersection
- 3 or complement: Complement
- 4 or filePattern: FilePattern
- 5 or intersectionCount: IntersectionCount

For more information about how each of these rules operate on input files, see Amazon Lumberyard Asset List Comparison Operations (p. 380).

Note

The intersectionCount comparison type can't be combined with any other comparison type as part of a rule list.

Type: Multi-value argument

Required: Yes

--filePatternType

The type of file pattern matching to use on the provided file patterns. Valid values are:

- 0: Perform wildcard matching - the * character will match any number of characters
- 1: Perform regular expression matching
Type: Multi-value argument. The number of parameters for the --filePatternType argument must match the number of FilePattern arguments to the --comparisonType argument.

Required: No

--filePattern

The file patterns to use for building the list of files that will be compared by the corresponding --comparisonType. The patterns are interpreted according to the corresponding --filePatternType parameter.

Type: Multi-value argument. The number of parameters for the --filePattern argument must match the number of FilePattern arguments to the --comparisonType argument.

Required: No

--allowOverwrites

Allow overwriting of existing comparison rules files. By default, existing files will not be overwritten.

Type: Flag

Required: No

Examples

Example Generate a delta and filter for XML files

Generate a comparison rules file which produces a delta comparison, and then filters the results to include only XML files:

```
Bin64\vc141\AssetBundlerBatch.exe comparisonRules --comparisonRulesFile deltaFilterXML.rules
^  --comparisonType delta,filePattern ^
  --filePatternType 0 ^
  --filePattern "*xml"
```

Comparisons - compare

The compare command is used to take pairs of asset lists as input, perform a comparison operation, and write the result of the comparison as a new asset list. See Amazon Lumberyard Asset List Comparison Operations (p. 380) for details on comparison operations.

Options

--comparisonRulesFile

The comparison rules file to load rules from. Comparisons from the rules file will be performed before any other comparisons given as arguments, and are evaluated in the order that the rules file was created with.

Type: Single-value argument

Required: No

--comparisonType

The comparison types to apply to the input files. The first --comparisonType parameter is applied to the first arguments of --firstAssetListFile and --secondAssetListFile, the second comparison is applied to the second parameters of those arguments, and so on.
Valid values are:
• 0 or `delta`: Delta comparison
• 1 or `union`: Union
• 2 or `intersection`: Intersection
• 3 or `complement`: Complement
• 4 or `filePattern`: FilePattern
• 5 or `intersectionCount`: IntersectionCount

For more information about how each of these rules operate on input files, see Amazon Lumberyard Asset List Comparison Operations (p. 380).

**Note**
The `intersectionCount` comparison type can't be combined with any other comparison type.

*Type:* Multi-value argument  
*Required:* Yes

---

**--filePatternType**

The type of file pattern matching to use on the provided file patterns. Valid values for this argument are:
• 0: Perform wildcard matching - the `*` character will match any number of characters
• 1: Perform regular expression matching

*Type:* Multi-value argument. The number of parameters for the `filePatternType` argument must match the number of `FilePattern` comparison arguments to the `comparisonType` argument.  
*Required:* No

---

**--filePattern**

The file patterns to use for matching asset file paths from inputs. The patterns are interpreted according to the corresponding `filePatternType` parameter. The first pattern is used with the first occurrence of the `FilePattern` comparison type, the second with the second occurrence, and so on.

*Type:* Multi-value argument. The number of parameters for the `filePattern` argument must match the number of `FilePattern` comparison arguments to the `comparisonType` argument.  
*Required:* No

---

**--firstAssetListFile**

The files to use as the first set of inputs for comparison.

*Type:* Multi-value argument. The number of parameters for the `firstAssetListFile` argument must match the number of arguments to the `comparisonType` argument.  
*Required:* No

---

**--secondAssetListFile**

The files to use as the second set of inputs for comparisons which require a second input file. This argument isn't used for the `FilePattern` or `intersectionCount` comparison types.

*Type:* Multi-value argument. The number of parameters for the `secondAssetListFile` argument must match the number of non-`FilePattern` arguments to the `comparisonType` argument.
--output

The output files for the result of each performed comparison. Output files can be a file, or a variable passed from another comparison. Variables start with the $ character. For more about variables, see Amazon Lumberyard Asset List Comparison Operations (p. 380).

Required: No

--print

This argument behaves differently depending on whether it's given as a flag or has a parameter list.

- **Flag (no arguments):** Prints the final comparison result to the console.
- **With arguments:** Prints the contents of each argument to the console after comparisons complete. Arguments can either be files or variables.

Required: No

--allowOverwrites

Allow overwriting of existing output files. By default, existing files will not be overwritten.

Type: Flag

Required: No

Examples

Example Compare to generate a delta

Generate a new asset list `deltaAssetList.assetlist` by taking the files which appear in either `firstAssetList_pc.assetlist` and `secondAssetList_pc.assetlist`, but not both:

```
Bin64vc141\AssetBundlerBatch.exe compare --comparisonType delta ^
   --firstAssetFile firstAssetList_pc.assetlist ^
   --secondAssetFile secondAssetList_pc.assetlist ^
   --output deltaAssetList.assetlist
```

Example Compare based on file path matching

Generate a new asset list `filePatternAssetList.assetlist` that contains only XML files from the `assetList_pc.assetlist` file:

```
Bin64vc141\AssetBundlerBatch.exe compare --comparisonType filePattern ^
   --filePatternType 0 ^
   --filePattern "*.xml" ^
   --firstAssetFile assetList_pc.assetlist ^
   --output filePatternAssetList.assetlist
```

Example Count intersection across multiple asset lists

Use `intersectionCount` on `engine_pc.assetlist`, `game_pc.assetlist`, and `patch_pc.assetlist` to print out assets which appear 2 times or more between any of these asset lists:
Bin64\vc141\AssetBundlerBatch.exe compare --comparisonType intersectionCount ^
--firstAssetFile engine_pc.assetlist,game_pc.assetlist,patch_pc.assetlist ^
--print

**Bundle settings - bundleSettings**

The `bundleSettings` command is used to manage bundle settings file, configuration files that let you store commonly-used bundle configurations for easy reuse and automation.

**Options**

--- `bundleSettingsFile`

The bundle settings file to be modified when running this command. If this file already exists, only those settings which are specified by the command invocation are changed.

*Type:* Single-value argument

*Required:* Yes

--- `assetListFile`

Sets the asset list file to use in bundle generation.

*Type:* Single-value argument

*Required:* No

--- `outputBundlePath`

Sets the location where the generated asset bundle is written to. Asset bundles use the `.pak` file extension.

*Type:* Single-value argument

*Required:* No

--- `bundleVersion`

Sets the bundle format version to use in generation. The only allowed value is 1.

*Type:* Single-value argument

*Required:* No

--- `maxSize`

Sets the maximum allowed size for individual bundles, in MB. If any generated bundle is larger than the maximum size, it will be split into smaller bundles and named accordingly.

*Type:* Single-value argument

*Required:* No

--- `platform`

The platforms to update the bundle settings for. Defaults to the project's enabled platforms, defined in `AssetProcessorPlatformConfig.ini`. Valid platform names can be found in the platform configuration file or as folder names under `dev\Cache\ProjectName`.

*Type:* Multi-value argument

*Required:* No
**--print**

Prints the contents of the bundle settings file to the console after modifying all values.

*Type: Flag*

*Required: No*

### Examples

The following examples assume that these platforms are enabled for the project: pc

#### Example Set default max bundle size and asset list for PC

Create a bundler settings file `defaults_pc.bundlesettings` for PC with the maximum bundle size set to 1024MB and the `allAssets_pc.assetlist` asset list as its input:

```
Bin64vc141\AssetBundlerBatch.exe bundleSettings --bundleSettingsFile defaults.bundlesettings ^
  --maxSize 1024 ^
  --assetListFile allAssets.assetlist ^
  --platforms pc
```

### Asset bundles - bundles

The `bundles` command is used to generate the final bundle (.pak) files that contain all of the assets from an asset list. Bundles can't be modified once created, only regenerated.

#### Options

**--bundleSettingsFile**

The bundle settings file to be loaded. If arguments are provided which would override the settings file, the arguments override the settings file.

*Type: Single-value argument*

*Required: No*

**--assetListFile**

The asset list file to use in bundle generation.

*Type: Single-value argument*

*Required: No*

**--outputBundlePath**

The location where the generated asset bundle is written to. Asset bundles use the .pak file extension.

*Type: Single-value argument*

*Required: No*

**--bundleVersion**

The bundle format version to use in generation. The only allowed value is 1.

*Type: Single-value argument*
Required: No

--maxSize

The maximum allowed size for individual bundles, in MB. If any generated bundle is larger than the maximum size, it will be split into smaller bundles and named accordingly.

Type: Single-value argument

Required: No

--platform

The platforms to generate bundles for. Defaults to the project's enabled platforms, defined in AssetProcessorPlatformConfig.ini. Valid platform names can be found in the platform configuration file or as folder names under dev\Cache\ProjectName.

Type: Multi-value argument

Required: No

--allowOverwrites

Allow overwriting of existing bundle files. By default, bundles are not overwritten.

Type: Flag

Required: No

Examples

In the following examples, assume that the current project has enabled the platforms pc, ios, and es3.

Example Create a bundle for PC using a settings file

Create a assets_pc.pak bundle for PC, using the defaults_pc.bundlesettings file:

Bin64vc141\AssetBundlerBatch.exe bundles --outputBundlePath assets.pak --bundleSettingsFile defaults.bundlesettings --platform pc

Example Create bundles for all platforms

Create bundles for all of a project's enabled platforms, using the allAssets_pc.assetlist, allAssets_ios.assetlist, and allAssets_es3.assetlist files:

Bin64vc141\AssetBundlerBatch.exe bundles --outputBundlePath assets.pak --maxSize 512 -- assetListFile allAssets.assetlist

Bundle from seed - bundleSeed

The bundleSeed command is used to generate bundles directly from seeds and their dependencies, without the use of an intermediate asset list. No other files besides the required seeds are used as input, and only bundle files are produced as output.

Options

--addSeed

The seeds to be used in bundle file generation. All asset dependencies of these seeds are included in the bundle as well. Argument parameters should be given as cache-relative paths to pre-processed assets.
Type: Multi-value argument

Required: Yes

--bundleSettingsFile

The bundle settings file to be loaded. If arguments are provided which would override the settings file, the arguments override the settings file.

Type: Single-value argument

Required: No

--outputBundlePath

The location where the generated asset bundle is written to. Asset bundles use the .pak file extension.

Type: Single-value argument

Required: No

--bundleVersion

The bundle format version to use in generation. The only allowed value is 1.

Type: Single-value argument

Required: No

--maxSize

The maximum allowed size for individual bundles, in MB. If any generated bundle is larger than the maximum size, it will be split into smaller bundles and named accordingly.

Type: Single-value argument

Required: No

--platform

The platforms to update the bundle settings for. Defaults to the project's enabled platforms, defined in AssetProcessorPlatformConfig.ini. Valid platform names can be found in the platform configuration file or as folder names under dev\Cache\ProjectName.

Type: Multi-value argument

Required: No

--allowOverwrites

Allow overwriting of existing bundle files. By default, bundles are not overwritten.

Type: Flag

Required: No

Examples

Example Regenerate bundle for a seed

Regenerate the bundle processed.pak for the example.cgf asset and all of its dependencies, with a maximum size of 512MB.

Bin64vc141\AssetBundlerBatch.exe bundleSeed --addSeed example.cgf --outputBundlePath processed.pak --maxSize 512 --allowOverwrites
Customize FBX asset export with FBX Settings

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Meshes, actors, PhysX colliders, and motions created in third-party content creation tools must be exported to a runtime format for your project. To export your assets to Lumberyard, save the assets from your content application as .fbx files. Then, place the .fbx files in one of the asset directories of your project. Lumberyard uses .fbx as an intermediate file format because most modeling and animation applications can read and create .fbx files.

Topics
- FBX Settings introduction (p. 409)
- FBX Settings export properties (p. 409)
- FBX Settings mesh export (p. 436)
- FBX Settings actor export (p. 437)
- FBX Settings motion export (p. 438)
- FBX Settings PhysX export (p. 439)
- Multiple UV sets for meshes and actors (p. 441)
- FBX soft naming conventions (p. 443)

FBX Settings introduction

When you place .fbx files in an asset directory in your project, Asset Processor detects the new or modified files, determines the contents of the files, and then exports the data using basic settings. However, .fbx files can be complex and might contain data that is necessary for an artist, animator, or designer, but not necessary for a runtime asset. Data in the .fbx file might require special handling such as higher precision vertex data or a coordinate space change. With FBX Settings, you can specify what data in the .fbx file to export, and how the data should be processed by Asset Processor.

When you customize export properties and add modifiers with FBX Settings, a corresponding .assetinfo file containing the settings is created for the .fbx file. The .fbx file is not changed. When Asset Processor exports the .fbx file, it uses the settings in the .assetinfo file to generate the runtime asset.

You can find a sample .fbx files in the lumberyard_version\dev\SamplesProject\Objects\Tutorials\Fbx directory, or type fbx into the search field at the top of Asset Browser to show .fbx files in your current project.

FBX Settings export properties

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Topics

- FBX Settings overview (p. 410)
- Modify FBX Settings (p. 411)
- FBX Settings Meshes tab (p. 413)
- FBX Settings Actors tab (p. 419)
- FBX Settings Motions tab (p. 426)
- FBX Settings PhysX tab (p. 429)

FBX Settings overview

With FBX Settings you can customize how Asset Processor exports your .fbx files as runtime assets. FBX Settings presents options based on the contents of the selected .fbx file.

Your .fbx files can contain any number of assets. You can create a single .fbx file that contains multiple characters, with LoDs, and animations, or a single .fbx file containing all the vegetation assets.
required for a level. When these assets are processed they appear as children of the .fbx file in Asset Browser.

### Modify FBX Settings

1. In Lumberyard Editor, in Asset Browser, select the .fbx file to modify.

   **Tip**
   
   Enter `fbx` into Asset Browser's search bar to find .fbx files.

2. Right-click the .fbx file and then choose Edit Settings.
3. There are four possible tabs in **FBX Settings**. The available tabs depend on the contents of the `.fbx` file:

- **Meshes** - In the **Meshes** tab, you can create groups and modify export settings for meshes. Exported mesh data is stored in a `.cgf` file.

  The **Meshes** tab is available if the `.fbx` file contains meshes.

- **Actors** - In the **Actors** tab, you can create groups and modify export settings for actors. An actor has a skeleton and a skinned mesh, but is not necessarily a character. Any asset containing at least one bone is an actor. Actor data is stored in a `.actor` file.

  The **Actors** tab is available if the `.fbx` file contains at least one bone.

- **Motions** - In the **Motions** tab, you can modify export settings for animation. Animation data is stored in a `.motion` file.

  The **Motions** tab is available if the `.fbx` file contains at least one bone with animated channels.

- **PhysX** - In the **PhysX** tab, you can create groups and modify export settings for PhysX collider assets. You set properties to automatically fit primitive or generate convex collider assets based on meshes contained in the `.fbx` file. PhysX collider data is stored in a `.pxmesh` file.

  The **PhysX** tab is available if the `.fbx` file contains meshes.

  **Note**

  To use PhysX collider assets, you must have the PhysX Gem enabled in your project.

4. The **Meshes**, **Actors**, and **PhysX** tabs can have multiple groups, each with their own modifiers. Each group creates separate runtime asset files. To add groups, choose the **Add another...** button at the top of the tab. To add a modifier to a group, choose the **Add Modifier** button in the **Modifiers** section of the group and select a modifier from the list.

5. The **Motions** tab exports animations individually and creates a `.motion` file for each animation. To add motions, choose the **Add another motion** button at the top of the tab. To add a modifier to a motion, choose the **Add Modifier** button in the **Modifiers** section of the motion, and select a modifier from the list.

6. To remove a modifier, motion, or group, choose the **X** button for the entry.

7. Choose **Update** to apply your changes. A `.assetinfo` file containing your modified FBX settings is generated or the existing `.assetinfo` file is updated. **Asset Processor** automatically exports the data from your `.fbx` files. Some `.fbx` files might take longer to process depending on file size, complexity, and selected options and modifiers.
8. Review the status for errors or a success message. To return to the settings, choose OK.

The exported runtime assets appear as children of the .fbx file in Asset Browser.

**FBX Settings Meshes tab**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

All meshes in the .fbx file are processed to a single runtime asset (.cgf) by default. In the Meshes tab, you can create mesh groups containing specific meshes from the .fbx file. Each Mesh group produces its own .cgf file. The processed runtime assets appear in Asset Browser as children of the .fbx file.

**Contents**

- Meshes tab properties (p. 414)
- Cloth modifier (p. 415)
- Comment modifier (p. 416)
- Level of Detail modifier (p. 416)
- Material modifier (p. 416)
- Mesh (Advanced) modifier (p. 417)
- Origin modifier (p. 418)
- Touch Bending modifier (p. 418)
- CryPhysics Proxy modifier (p. 419)
Meshes tab properties

Add another mesh

Add a mesh group to process. Each mesh group can contain one or more meshes from the .fbx file. Each mesh group produces a .cgf file.

Name mesh

Enter a name for the mesh group. This is the name of the .cgf file containing the processed meshes. The .cgf file appears in Asset Browser as a child of the .fbx file.

Select meshes

Specify the meshes to process from the .fbx file for this mesh group. Choose the Hierarchy icon to see a list of meshes found in the .fbx file. Select meshes from the list to include them in the mesh group and process as visible render meshes. If the .fbx contains meshes that are intended to be used as PhysX collider assets, you should exclude them from the Mesh group by deselecting them in this list.

Add Modifier

Modifiers add specialized options for processing assets. Choose the Add Modifier button to see a list of available modifiers:

- Cloth
- Comment
- CryPhysics Proxy
- Level of Detail
- Material
- Mesh (Advanced)
- Origin
- Touch Bending
Note
Some modifiers are not available unless the gem that provides the modifier is enabled in your project.

Cloth modifier

Add NVIDIA Cloth data to a selected mesh to simulate cloth physics.

Note
Each mesh in the Mesh group to simulate as cloth requires its own Cloth modifier. When a Cloth modifier is added to a mesh group, the Merge Meshes property in the Mesh (Advanced) modifier will be ignored and treated as disabled. A cloth mesh needs to be processed independently and cannot be merged with other meshes.

For more information, see Simulate cloth with NVIDIA Cloth (p. 2857).

Select Cloth Mesh

Select the mesh to have cloth data applied and simulate as a cloth object.

Note
For information on the Inverse Masses, Motion Constraints, and Backstop properties below, see Per vertex properties for cloth (p. 2866).

Inverse Masses

Select a vertex color stream to apply per vertex inverse mass data for cloth simulation. If no vertex color stream is selected, an inverse mass value of 1.0 is assigned to all vertices in the cloth mesh.

Inverse Masses Channel

Select the channel in the vertex color stream that contains inverse mass data.

Motion Constraints

Select a vertex color stream to apply per vertex motion constraints data for cloth simulation. If no vertex color stream is selected, a motion constraint value of 1.0 is assigned to all vertices in the cloth mesh.

Motion Constraints Channel

Select the channel in the vertex color stream that contains motion constraints data.

Backstop

Select a vertex color stream to apply per vertex backstop data for cloth simulation. If no vertex color stream is selected, backstop is disabled for the cloth mesh.
Backstop Offset Channel

Select the channel in the vertex color stream that contains backstop offset data.

Backstop Radius Channel

Select the channel in the vertex color stream that contains backstop radius data.

Comment modifier

Add a comment to the file. You can add a comment about changes made to the .fbx file for tracking purposes or notes on export options, for example. Comments don't affect how files are processed. Multiple comment modifiers can be added to a mesh group.

Level of Detail modifier

Add level of detail (LoD) meshes to the Mesh group.

1. Choose the + button to add an LoD.
2. Choose the Trash Can button to remove an LoD.
3. Choose the Hierarchy button to specify the meshes to include in the LoD.

LoDs are optimized meshes with progressively lower polygon counts, fewer and smaller textures, and simplified materials. The farther an entity is from the camera, the less detail is required from the meshes that make up the entity. As the entity moves farther from the camera, it swaps to a lower LoD.

In addition to the base static mesh, you can specify up to five LoDs which are numbered [1] to [5], with [1] being the highest level of detail. As the entity moves further from the camera, it will transition from the base mesh to LoD 1, and progressively through the LoDs the further it moves from the camera. LoDs are not required. Creating LoDs, however, is recommended because they help get the best performance and visual fidelity across a range of platforms with different hardware capabilities.

Note

When you author the mesh in your 3D application, you can add _lod1, _lod2, _lod3, _lod4, _lod5 as suffixes to your mesh names to automatically add a Level of Detail modifier and assign the meshes to appropriate LoDs. _lod1 is mapped to [0], _lod2 is mapped to [1], and so on.

Material modifier

The Material modifier helps automatically manage the contents of the .mtl file that corresponds to the mesh group when mesh assets are updated.
A material is a combination of shaders and properties that define the surface of a mesh. Materials contain shader and texture assignments, settings for shader properties such as smoothness, opacity, emissive color, etc., and if necessary, a physics material assignment that defines physical properties such as friction. A mesh group processed with a file named `myfile.cgf` has a corresponding material file named `myfile.mtl`.

When a mesh group is processed, **Asset Processor** generates a material file (.mtl) containing a list of materials and their properties for the mesh group. A mesh can have multiple materials, and a mesh group can have multiple meshes, so the .mtl file might contain several materials even if the asset seems simplistic visually.

**Update materials**

When enabled, updates the texture map file names in the .mtl file to match the texture map names in defined in the .fbx file.

**Remove unused materials**

When enabled, removes materials that are present in the .mtl file that are not defined in the .fbx file.

**Mesh (Advanced) modifier**

![Mesh Modifier](image)

The **Mesh (Advanced)** modifier adds advanced mesh processing features such as a setting for vertex precision, which can save memory, and a setting to choose a vertex color stream to include in the processed mesh.

**Vertex Precision**

Select the precision of the vertex data for the mesh group. **16-bit** floats have a range of -65,500 to 65,500 and 3 significant decimal places. Using **16-bit** precision results in a much smaller file size for processed meshes, but vertices in the resulting .cgf file might shift slightly from their positions in the .fbx file. This position shift might be noticeable on large meshes, meshes with precise detail, or meshes that are placed far from the origin. Choose **32-bit** precision for larger maximum values and more significant decimal places if the vertices appear to have shifted in the runtime asset.

**Important**

32-bit precision vertices consume more resources and might result in a loss of performance on some platforms. Some platforms have native support for 16-bit precision which can offer improved performance. Check the capabilities of your target platform to determine which precision offers the best results.

**Merge Meshes**

When enabled, combines all sub-meshes in the mesh group into a single mesh for optimization.

**Use Custom Normals**

Enable this property to use custom normals, otherwise, **Asset Processor** generates averaged normals.

Normals are vertex attributes that define the surface direction of your meshes. Normals can be customized in third-party modeling tools to make a mesh appear faceted, create hard edges.
between surfaces, or smooth the appearance of a surface. Custom normals can be included in .fbx files.

**Vertex color stream**

If the mesh for this **Mesh group** contains a vertex color stream, it can be selected from this list to be processed.

Vertex color streams contain per vertex color data that can be referenced by materials. Vertex color streams are also often used for tagging meshes with arbitrary data such as the inverse mass value used in cloth simulation. Because of this, a mesh might have multiple vertex color streams. Be sure to select a vertex color stream intended to be referenced by materials if multiple streams exist.

**Origin modifier**

<table>
<thead>
<tr>
<th>Origin</th>
<th>Relative Origin Node</th>
<th>X 0.0</th>
<th>Y 0.0</th>
<th>Z 0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Translation</td>
<td>P 0.0</td>
<td>R 0.0</td>
<td>Y 0.0</td>
</tr>
<tr>
<td></td>
<td>Scale</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Change the position (translation), orientation (rotation), and scale of a mesh relative to how it was authored.

**Relative Origin Node**

Select the transform relative to which the mesh is processed. By default, the mesh origin is placed at the scene position 0, 0, 0 in the .fbx file.

**Translation**

Sets the position offset of the processed mesh.

**Rotation**

Sets the orientation offset of the processed mesh in degrees.

**Scale**

Sets the scale offset of the processed mesh.

**Touch Bending modifier**

**TouchBending**

- Select root bone
- Proximity Trigger Mesh(es) 0 of 4 proximity meshes selected
- Stiffness 0.5
- Damping 0.5
- Thickness[m] 0.01

The **Touch Bending** modifier sets up mesh assets for touch bending. Touch bending is a collision effect, typically used on vegetation assets such as plants or tall grass, that causes the asset to bend away from an entity that brushes against it. You can use this modifier to create a field of wheat, for example, where the wheat parts and bends as a player character passes through.
Touch bendable assets require a rig similar to an actor. For more information on creating touch bendable assets, see Adding Touch (Collision) Bending Effects (p. 1302).

**Select root bone**

Specify the root bone of the touch bendable mesh.

**Proximity Trigger Mesh(es)**

Choose the Hierarchy button to select collision meshes that trigger the touch bending effect. Multiple trigger meshes can be selected, however, each additional trigger mesh decreases performance. A new PhysicsNoDraw material with a NoCollide type is created for the mesh(es).

**Stiffness**

Set a stiffness value between 0.0 and 1.0 for all branches to define how easily the asset bends.

**Damping**

Set a damping value between 0.0 and 1.0 for all branches to define how quickly the asset returns to its rest position after a collision.

**Thickness**

Set a thickness value for all branches to define the amount of bending. Thickness is determined as the radius of a cylinder in meters. Most often a small decimal number less than 1.0 is required. Valid values range from 0.00001 to infinity.

**CryPhysics Proxy modifier**

![CryPhysics Proxy](image)

**Important**

The legacy physics system will be deprecated in a future Lumberyard release. Use the PhysX system instead.

Select meshes to use as proxy physics meshes for the legacy physics system.

**Physics meshes**

Choose the Hierarchy icon to specify the meshes to use for physics proxies from the .fbx file. Physics proxies are meshes that encapsulate render geometry (for example, hit detection or physics collision) and are optimized with a low polygon count for better performance. Primitives such as a cube, sphere or capsule are best for optimal physics performance.

**Note**

If your .fbx file includes a mesh node with the suffix _phys, the mesh node automatically adds a new Physics Proxy modifier.

**FBX Settings Actors tab**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Actors are assets with at least one bone and can contain one or more skinned meshes. By default, all actors in the `.fbx` file are processed. However, you can manually exclude individual actors within your `.fbx` file. You can also process multiple actors from a single `.fbx` file. Each Actor group produces its own `.actor` file. The processed runtime assets appear in Asset Browser as children of the `.fbx` file.

Contents

- Actors tab properties (p. 420)
- Cloth modifier (p. 421)
- Comment modifier (p. 422)
- Coordinate system change modifier (p. 422)
- Level of Detail modifier (p. 423)
- Material modifier (p. 423)
- Mesh modifier (p. 424)
- Scale actor modifier (p. 424)
- Skeleton optimization modifier (p. 424)
- Skin modifier (p. 425)
- Tangents modifier (p. 425)

Actors tab properties

Add another actor

Add an Actor group to process. Each Actor group produces a `.actor` file.

Name actor

Enter a name for the Actor. This is the name of the `.actor` file that appears in Asset Browser as a child of the `.fbx` file.
Select root bone

Select the root bone of the skeleton hierarchy. The top parent bone of the skeleton is automatically selected as the default bone. All descendant bones of the root bone are processed.

Select base meshes

Specify the meshes to process from the .fbx file for this Actor group. Choose the Hierarchy icon to see a list of meshes found in the .fbx file. Select meshes from the list to include them in the Actor group and process as visible render meshes. If the .fbx contains meshes that are intended to be used as PhysX collider meshes, or meshes that are not skinned to the skeleton hierarchy being processed, you should exclude them from the Actor group by deselecting them in this list.

Note

If you have defined level of detail meshes for your actor by appending _lod0 through _lod5 to meshes in the .fbx file, the _lod0 meshes are automatically selected for the Select base meshes property.

Add Modifier

Modifiers add additional specialized options for processing assets. Choose the Add Modifier button to see a list of available modifiers:

- Cloth
- Comment
- Coordinate system change
- Material
- Mesh
- Scale actor
- Skeleton optimization
- Skin
- Tangents

Note

Some modifiers are not be available unless the gem that provides the modifier is enabled in your project.

Cloth modifier

Add NVIDIA Cloth data to a selected mesh to simulate cloth physics.

Note

Each mesh in the Actor group to simulate as cloth requires its own Cloth modifier.

For more information, see Simulate cloth with NVIDIA Cloth (p. 2857).
Select Cloth Mesh

Select the mesh to have cloth data applied and simulate as a cloth object.

**Note**
For information on the Inverse Masses, Motion Constraints, and Backstop properties below see Per vertex properties for cloth (p. 2866).

**Inverse Masses**

Select a vertex color stream to apply per vertex inverse mass data for cloth simulation. If no vertex color stream is selected, an inverse mass value of 1.0 is assigned to all vertices in the cloth mesh.

**Inverse Masses Channel**

Select the channel in the vertex color stream that contains inverse mass data.

**Motion Constraints**

Select a vertex color stream to apply per vertex motion constraints data for cloth simulation. If no vertex color stream is selected, a motion constraint value of 1.0 is assigned to all vertices in the cloth mesh.

**Motion Constraints Channel**

Select the channel in the vertex color stream that contains motion constraints data.

**Backstop**

Select a vertex color stream to apply per vertex backstop data for cloth simulation. If no vertex color stream is selected, backstop will be disabled for the cloth mesh.

**Backstop Offset Channel**

Select the channel in the vertex color stream that contains backstop offset data.

**Backstop Radius Channel**

Select the channel in the vertex color stream that contains backstop radius data.

**Comment modifier**

Add a comment to the file. You can add a comment about changes made to the .fbx file for tracking purposes or notes on export options, for example. Comments don't affect how files are processed and multiple comment modifiers can be added to a mesh group.

**Coordinate system change modifier**

Modify the coordinate system of the actor. Third-party content creation applications use varying coordinate systems with content applications often rotating the direction of the forward axis. The Facing direction property can be set to rotate the actor 180 degrees around its up axis to account for this difference. The rotation is applied when the asset is processed and the .fbx file remains unchanged.
Level of Detail modifier

The Level of detail (LoD) modifier is added to the Actor group automatically when LoDs are found in the .fbx file. Choose the Hierarchy button to verify or modify the meshes and bones assigned to each LoD. Actors support up to 5 levels of detail in addition to the base mesh(es).

Note
The Level of detail modifier for Actor groups cannot be added or removed manually and does not appear in the modifier list. The Level of detail modifier is automatically added and configured when actor LoDs are found in an .fbx file.

LoDs are optimized assets with progressively lower polygon counts, fewer and smaller textures, and simplified materials. An Actor LoD can also have a simplified skeleton where some leaf bones have been removed from the skeleton. The farther an entity is from the camera, the less detail is required from the actor contained in the entity. As the entity moves farther from the camera, it swaps to a lower actor LoD.

In addition to the base actor mesh, you can create up to five LoDs for actors which are numbered [1] to [5], with [1] being the highest level of detail. As the actor moves away from the camera, it will transition from the base mesh to LoD 1, and then progressively through the LoDs the further it moves from the camera. LoDs are not required. Creating LoDs, however, is recommended because they help get the best performance and visual fidelity across a range of platforms with different hardware capabilities.

Note
When you author the actor in your 3D application, you must add _lod1, _lod2, _lod3, _lod4, _lod5 as suffixes to your mesh names to automatically add a Level of Detail modifier and assign the meshes to appropriate LoDs. _lod1 is mapped to LOD 1, _lod2 is mapped to LOD 2, and so on.

Important
The meshes for the base mesh, and the corresponding skeleton are assigned in the Select base meshes and Select root bone properties of the Actor group automatically. Meshes labeled _lod1 through _lod5 and their skeleton hierarchies are assigned to corresponding LoDs in the Level of Detail modifier.

For more information, see Using Actor LODs to Optimize Game Performance (p. 1498).

Material modifier

The Material modifier helps automatically manage the contents of the .mtl file that corresponds to the Actor group when actor assets are updated.

A material is a combination of shaders and properties that define the surface of a mesh. Materials contain shader and texture assignments, settings for shader properties such as smoothness, opacity, emissive color, etc., and if necessary, a physics material assignment that defines physical properties such as friction. An Actor group processed with a file named myfile.actor has a corresponding material file named myfile.mtl.
When an **Actor group** is processed, **Asset Browser** generates a material file (.mtl) containing a list of materials and their property settings for the mesh group. A mesh can have multiple materials, and an **Actor group** can have multiple meshes, so the .mtl file might contain several materials even if the asset seems simplistic visually.

**Update materials**

When enabled, updates the texture map file names in the .mtl file to match the texture map names defined in the .fbx file.

**Remove unused materials**

When enabled, removes materials that are present in the .mtl file that are not defined in the .fbx file.

**Mesh modifier**

<table>
<thead>
<tr>
<th>Mesh</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex color stream</td>
<td>No vertex colors</td>
</tr>
<tr>
<td>Vertex color mode</td>
<td>32 bit (8 bits per channel)</td>
</tr>
</tbody>
</table>

Set a vertex color stream and its precision for the Actor's visible render mesh.

**Vertex color stream**

If the mesh for this **Actor group** contains a vertex color stream, it can be selected from this list to be processed.

Vertex color streams contain per vertex color data that can be referenced by materials. Vertex color streams are also often used for tagging meshes with arbitrary data such as the inverse mass value used in cloth simulation. Because of this, a mesh might have multiple vertex color streams. Be sure to select a vertex color stream intended to be referenced by materials if multiple streams exist.

**Vertex color mode**

Set the vertex color precision to either 32 bit (8 bits per channel) or 128 bit (32 bits per channel).

Vertex color streams contain four channels: red, green, blue, and alpha. Setting the mode to 32 bit vertex color precision saves significant memory over 128 bit vertex color precision at a loss of dynamic range. If you are targeting a platform with memory constraints, using 32 bit precision is a very useful optimization. If you are targeting a platform that supports high dynamic range (HDR) display, 128 bit precision offers much greater color fidelity.

**Scale actor modifier**

<table>
<thead>
<tr>
<th>Scale actor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale factor</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The **Scale factor** modifier sets a uniform scale for the **Actor group**. This setting is useful if your assets are created in an application that uses a different base standard unit of measurement than Lumberyard.

**Skeleton optimization modifier**

<table>
<thead>
<tr>
<th>Skeleton Optimization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Skeleton LOD</td>
<td></td>
</tr>
<tr>
<td>Server Skeleton Optimize</td>
<td></td>
</tr>
<tr>
<td>Critical bones</td>
<td>0 of 35 bones selected</td>
</tr>
</tbody>
</table>
The **Skeleton optimization** modifier automatically optimizes Actor skeletons for client and server scenarios.

**Auto Skeleton LoD**

When enabled, unskinned bones (bones not weighted to any vertices), are optimized out of the client side skeleton hierarchy.

**Server Skeleton Optimize**

When enabled, bones that do not have colliders attached are optimized out of the server side skeleton hierarchy.

**Critical bones**

Choose the **Hierarchy** button to select bones that should not be optimized out of the hierarchy. Your actor might have bones that have no skin weights such as attachment bones for weapons and accessories, for example, that you do not want optimized out of the hierarchy.

---

**Skin modifier**

<table>
<thead>
<tr>
<th>Max weights per vertex</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight threshold</td>
<td>0.001</td>
</tr>
</tbody>
</table>

The **Skin** modifier sets the maximum bones and minimum weight per vertex.

**Max weights per vertex**

The maximum number of bones that can influence a vertex. Values range from a minimum of 1 to a maximum of 4.

**Weight threshold**

The minimum bone weight per vertex. Weight values beneath this threshold are ignored during export. Values range from a minimum of 0 to a maximum of 0.01.

---

**Tangents modifier**

<table>
<thead>
<tr>
<th>Tangent space</th>
<th>MikkT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitangents</td>
<td>Orthogonal</td>
</tr>
<tr>
<td>Normalize</td>
<td>On</td>
</tr>
<tr>
<td>Uv set</td>
<td>0</td>
</tr>
</tbody>
</table>

The **Tangents** modifier generates or exports tangents and bitangents from your .fbx file. You should use the same tangent space you use to generate normal maps to achieve the desired appearance in Lumberyard.

Normal maps can make a surface appear as though it has more geometric surface detail than exists in the mesh. The normals in the map are in tangent space which points in the positive Z direction. Tangent space is local to the surface and is used to transform the normals in the normal map into world space so the material can be lit and rendered correctly on a deforming surface such as a character mesh. You can create tangents and bitangents in your third-party content application and save them as vertex attributes, then export the tangents with the **Tangents** modifier, or use the **Tangents** modifier to generate tangents when the mesh for this **Actor group** is processed.
Tangent space

Defines the tangent space that Lumberyard uses for an actor. Choose from the following options:

- **MikkT** – Uses MikkT to generate the tangents. MikkT is a widely used solution for generating tangent space that strives for consistency regardless of changes and optimizations to the source mesh. This is the default setting.
- **From FBX** – Export the tangents from the FBX file.
- **EMotion FX** – Use EMotion FX to calculate the tangents.

**Warning**
The **EMotion FX** option is for backward compatibility only and is not recommended.

Bitangents

Defines how to generate or read the bitangents.

- **Use from source** – Use the bitangents from the FBX file or the output from the tangent generator used (MikkT).
- **Orthogonal** – Generates the bitangents orthogonal to the tangent and normal. This is the default setting.

**Normalize**

When enabled, tangents and bitangents are normalized. **Normalize** is enabled by default.

UV set

The index of the UV set to generate tangents from. The default setting is 0, the first UV set.

FBX Settings Motions tab

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can process animation sequences from a single .fbx file as **Motions**. Each **Motion** produces its own .motion file. The processed runtime assets appear in **Asset Browser** as children of the .fbx file.

**Important**

Animation in your .fbx should be baked, that is, each animated channel should be keyed at every frame. Third-party animation applications might use functions and simulations to interpolate between keyframes and the results are not exported from your .fbx file unless the animation is baked into the animation channels.

For more information, see Animation Editor File Types (p. 1352).

Contents

- Motions tab properties (p. 427)
- Additive motion modifier (p. 427)
- Comment modifier (p. 428)
- Compression Settings modifier (p. 428)
- Coordinate system change modifier (p. 428)
- Motion range modifier (p. 429)
- Scale motion modifier (p. 429)

**Motions tab properties**

![Image of FBX export properties settings](image)

**Add another motion**

Add an animation to export as a .motion from the .fbx file.

**Name motion**

Enter a name for the motion. This is the name of the .motion file that appears in **Asset Browse** as a child of the .fbx file.

**Select root bone**

Select the root bone of the animated skeleton hierarchy. By default the top parent bone of the skeleton is selected as the root bone.

**Add Modifier**

Modifiers add specialized options for processing assets. Choose the **Add Modifier** button to see a list of available modifiers:

- Additive motion
- Comment
- Compression settings
- Coordinate system change
- Motion range
- Scale motion

**Additive motion modifier**

![Image of Additive motion modifier](image)
Export the animation as an additive motion. Additive motions can be layered on top of base motions without affecting the base motion functionality.

**Base Frame**

Specifies the number of the base frame that contains the reference pose.

For more information, see

**Comment modifier**

Add a comment to the file. You can add a comment about changes made to the .fbx file for tracking purposes or notes on export options, for example. Comments don't affect how files are processed and multiple comment modifiers can be added to a mesh group.

**Compression Settings modifier**

Reduce asset size by compressing animation. The **Compression settings** modifier sets tolerances for keyframe values on each transform type. If the change in a keyframe's value from the preceding keyframe is smaller than the tolerance value set in this modifier, the keyframe is removed.

**Max translation error tolerance**

Specify the maximum error tolerance allowed in translation. Valid values range from a minimum of 0 to a maximum of 0.1.

**Max rotation error tolerance**

Specify the maximum error tolerance allowed in rotation. Valid values range from a minimum of 0 to a maximum of 0.001.

**Max scale error tolerance**

Specify the maximum error tolerance allowed in scale. Valid values range from a minimum of 0 to a maximum of 0.01.

**Coordinate system change modifier**

Modify the coordinate system of the motion. Third-party content creation applications and game engines use varying coordinate systems with content applications often rotating the direction of the forward axis. The **Facing direction** property can be set to rotate the motion 180 degrees around its up axis to account for this difference. The rotation is applied when the asset is processed and the .fbx file remains unchanged.
Motion range modifier

Set the range of the animation to be exported from the .fbx file.

**Start frame**

Specify the start keyframe of the animation to export.

**End frame**

Specify the end keyframe of the animation to export.

Scale motion modifier

The **Scale factor** modifier sets a uniform scale for the **Motion**. This setting is useful if your assets are created in an application that uses a different base standard unit of measurement than Lumberyard.

FBX Settings PhysX tab

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

In the PhysX tab, you can create **PhysX Mesh groups** to process collider assets for PhysX. Collider assets can be triangle meshes, or generated as primitives or convex meshes based on meshes contained in the .fbx file. Multiple **PhysX Mesh groups** can be processed from a single .fbx file. Each **PhysX Mesh group** produces its own .pxmesh file. The processed runtime assets appear in Asset Browser as children of the .fbx file.

**Important**

There are many options for creating PhysX collider assets. The best options in a scenario depend on many factors including mesh complexity, how the collider asset is used, and whether the entity containing the collider is static (doesn't move), kinematic (animated), or dynamic (has a rigid body component). In general, primitive colliders offer the best simulation performance, but you might consider trading performance for precision in situations where collider assets that closely match a visible render mesh are desirable.

Contents

- PhysX tab properties (p. 430)
- Triangle mesh asset properties (p. 431)
- Convex asset properties (p. 432)
- Primitive asset properties (p. 433)
- Decomposition properties (p. 434)
• PhysX modifiers (p. 435)
• Comment modifier (p. 435)
• Origin modifier (p. 436)

PhysX tab properties

Add another PhysX mesh group to process as a PhysX collider asset (.pxmesh) from the .fbx file.

Name PhysX Mesh

Enter a name for the PhysX mesh. This is the name of the .pxmesh file that appears in Asset Browser as a child of the .fbx file.

Select meshes

Specify the meshes to process from the .fbx file for this PhysX mesh group. Choose the Hierarchy icon to see a list of meshes found in the .fbx file. Select meshes from the list to include them in the PhysX mesh group and process as PhysX collider assets.

Note
If your .fbx file includes mesh nodes with the suffix _phys, the meshes are automatically selected for processing in the PhysX mesh group.

Export As

The cooking method to apply to this PhysX mesh group. This setting exposes properties for the selected cooking method. The three options are:
• Triangle - Generate colliders composed of triangles based on the selected meshes.
• Convex - Generate convex hull colliders based on the selected meshes.
• Primitive - Fit primitive colliders based on the selected meshes.

Note
Triangle mesh colliders can only be used with static and kinematic entities. To use the PhysX collider asset with dynamic entities (entities that have a PhysX Rigid Body component), choose Convex or Primitive for the Export as property.
Triangle mesh asset properties

Triangle mesh colliders accurately reproduce the shape of the selected meshes, but cannot be used on dynamic entities. Triangle mesh colliders are most suitable for static environment entities that have complex shapes and require colliders that accurately resemble the visible render mesh shape.

<table>
<thead>
<tr>
<th>Triangle Mesh Asset Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge Meshes</td>
</tr>
<tr>
<td>Weld Vertices</td>
</tr>
<tr>
<td>Disable Clean Mesh</td>
</tr>
<tr>
<td>Force 32-Bit Indices</td>
</tr>
<tr>
<td>Suppress Triangle Mesh Remap Table</td>
</tr>
<tr>
<td>Build Triangle Adjacencies</td>
</tr>
<tr>
<td>Mesh Weld Tolerance</td>
</tr>
<tr>
<td>Number of Triangles Per Leaf</td>
</tr>
</tbody>
</table>

**Merge Meshes**

When enabled, all selected mesh nodes are merged into a single collider mesh. Otherwise, the selected mesh nodes are exported as separate shapes. It is usually more efficient to have a single collider mesh.

**Weld Vertices**

When enabled, mesh vertex welding is performed. **Disable clean mesh** must be disabled when **Weld Vertices** is enabled or vertex welding is not performed.

**Disable Clean Mesh**

When enabled, mesh cleaning is disabled. This makes cooking faster. **Disable clean mesh** must be disabled when **Weld Vertices** is enabled or vertex welding is not performed.

**Force 32-bit Indices**

When enabled, 32-bit indices are created for the collider asset regardless of triangle count.

**Suppress Triangle Mesh Remap Table**

When enabled, the face remap table is not created. This saves a significant amount of memory, but the SDK won’t be able to provide the remap information for internal mesh triangles returned by collisions, sweeps, or raycast hits.

**Build Triangle Adjacencies**

When enabled, triangle adjacency information is created. You can get the adjacency triangles for a given triangle from `getTriangle`.

**Mesh Weld Tolerance**

When **Weld Vertices** is enabled, this value is the distance within which vertices are welded. When **Weld Vertices** is disabled, this value defines the acceptance distance for mesh validation.

Having a clean, welded mesh is required to achieve the best possible performance. **Weld Vertices** uses a snap-to-grid approach. This approach truncates each vertex to an integer value using **Mesh Weld Tolerance**. Once these snapped vertices are produced, all vertices that snap to a given vertex on the grid are remapped to reference a single vertex. Following this, all triangle indices are remapped to reference this subset of clean vertices. Vertices do not have their positions modified; the snap-to-grid is only performed to identify nearby vertices.

The mesh validation approach uses the same snap-to-grid approach to identify nearby vertices. If more than one vertex snaps to a given grid coordinate, the distance between the vertices is checked.
to ensure it is greater than **Mesh Weld Tolerance**. If the vertices are within **Mesh Weld Tolerance**, a warning is emitted.

**Number of Triangles Per Leaf**

Set the mesh cooking hint for max triangles per leaf. Fewer triangles per leaf results in slower cooking speed and produces larger mesh sizes with better runtime performance. More triangles per leaf results in faster cooking speed and produces smaller mesh sizes with decreased runtime performance.

**Convex asset properties**

Convex hulls are generated colliders that can approximate the shape of the selected meshes. Convex hulls can be used with static, kinematic, and dynamic entities, and are often used on interactive entities such as weapons that require rigid body physics and a collider mesh that resembles the shape of the visible render mesh.

<table>
<thead>
<tr>
<th>Convex Asset Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area Test Epsilon</td>
<td>0.06</td>
</tr>
<tr>
<td>Plane Tolerance</td>
<td>0.0007</td>
</tr>
<tr>
<td>Use 16-bit indices</td>
<td>Off</td>
</tr>
<tr>
<td>Check Zero Area Triangles</td>
<td>Off</td>
</tr>
<tr>
<td>Quantize input</td>
<td>Off</td>
</tr>
<tr>
<td>Use Plane Shifting</td>
<td>Off</td>
</tr>
<tr>
<td>Shift Vertices</td>
<td>Off</td>
</tr>
<tr>
<td>Gauss Map Limit</td>
<td>32</td>
</tr>
<tr>
<td>Build GPU Data</td>
<td>Off</td>
</tr>
<tr>
<td>Decompose Meshes</td>
<td>Off</td>
</tr>
</tbody>
</table>

**Area Test Epsilon**

If the area of a triangle in the hull is below this value, the triangle is rejected. This test is performed only if **Check Zero Area Triangles** is set. Valid values range from a minimum of 0 to a maximum value of 100.

**Plane Tolerance**

This value is used during hull construction. When a new point is added to the hull, it is rejected when the point is closer to the hull than the **Plane Tolerance** value. The **Plane Tolerance** value is increased according to the hull size.

If the **Plane Tolerance** value is 0.0, all points are accepted when the convex hull is created. This can lead to edge cases where the new points are merged into an existing polygon, changing the polygon's plane equation slightly, resulting in failures during polygon merging phase in the hull computation.

We recommend the default value. However, if all points must be accepted, or large, thin convex hulls must be created, you can specify a lower value. Valid values range from a minimum of 0 to a maximum value of 100.

**Use 16-bit Indices**

When enabled, 16-bit triangle or polygon indices are generated. Otherwise, 32-bit vertex indices are generated.

**Check Zero Area Triangles**

Checks and removes triangles that are have a smaller area than the area value specified in **Area Test Epsilon**.
Quantize Input

Quantizes the input vertices using k-means clustering (p. 3283).

Use Plane Shifting

Enables plane shifting vertex limit algorithm. For more information, see the NVIDIA PhysX Geometry documentation.

Shift Vertices

When enabled, convex hull input vertices are shifted to be around the origin to provide better computation stability. It is recommended to provide input vertices around the origin; otherwise, use this property to improve numerical stability.

Gauss Map Limit

Specifies the vertex limit beyond which additional acceleration structures are computed for each convex mesh. Increase the limit to reduce memory usage. Computing the extra structures doesn't guarantee optimal performance. There is a per-platform break-even point below which the extra structures can negatively impact performance. Valid values range from a minimum of 0 to infinity.

Build GPU Data

When enabled, additional information required for GPU accelerated rigid body simulation is generated. This can increase memory usage and cooking time for convex meshes and triangle meshes. Convex hulls are generated with respect to GPU simulation limitations. The vertex limit is set to 64 and the per face vertex limit is set to 32.

Decompose Meshes

When enabled, the V-HACD algorithm is applied to split each node into approximately convex parts. Each part is exported as a convex collider using the properties configured above. Decompose Meshes enables Decomposition Properties which determine how the selected meshes are decomposed into approximately convex parts.

Primitive asset properties

Primitive colliders are simple parametric primitives (box, capsule, sphere) fit to the selected meshes and can be used with static, kinematic, and dynamic entities. Primitive colliders generally provide the best simulation performance, but might not closely match the shape of the visible render mesh. They are best suited for dynamic entities with simple meshes such as crates and barrels, as well as projectiles, triggers, and entities where colliders that accurately represent shape of the visible render mesh are not necessary.

Target Shape

Select a target shape to fit to the mesh: Box, Capsule, or Sphere. If Automatic is selected, then the algorithm determines which shape fits best.

Volume Term Coefficient

Specifies how aggressively the primitive fitting algorithm minimizes the volume of the fitted primitive. Valid values range from a minimum of 0 to a maximum of 0.002.

A value of 0 performs no volume minimization and is recommended for most meshes, especially meshes with moderate to high vertex counts.
For meshes that have low vertex counts, or vertices that are distributed mainly along the edges, the algorithm can sometimes produce sub-optimal results where the primitive touches the edges of the mesh, but not the faces. In these cases, the fitting can be improved by increasing the Volume Term Coefficient so that the fitting algorithm shrinks the volume of the primitive while minimizing its deviation from the mesh. A Volume Term Coefficient that is too high can shrink the primitive collider so that it’s occluded by the mesh.

**Decompose Meshes**

When enabled, the V-HACD algorithm is applied to split each node into approximately convex parts. Each part is exported as a primitive collider using the properties configured above. Decompose Meshes enables Decomposition Properties which determine how the selected meshes are decomposed into approximately convex parts.

**Decomposition properties**

Exporting a PhysX mesh as a convex or a primitive collider might not produce good results if the mesh’s shape is concave or doesn’t closely fit one of the primitive shapes. Exporting a PhysX mesh as a triangle mesh collider creates a collider that accurately resembles the original mesh, but won’t work with a dynamic entity. For these scenarios, Lumberyard supports approximate convex decomposition. Arbitrary meshes are broken down into approximately convex parts before processing each part through the asset pipeline individually.

Decomposing meshes has the advantage that each individual, nearly convex part, can be more accurately exported as a convex or primitive. Since the resulting asset doesn’t contain any triangle meshes, it can be used on dynamic entities.

For more information and illustrated examples of results, see the V-HACD library on GitHub.

<table>
<thead>
<tr>
<th>Decomposition Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Hulls</td>
<td>1024</td>
</tr>
<tr>
<td>Maximum Vertices Per Hull</td>
<td>64</td>
</tr>
<tr>
<td>Concavity</td>
<td>0.001</td>
</tr>
<tr>
<td>Resolution</td>
<td>100000</td>
</tr>
<tr>
<td>Advanced</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>Voxel-based</td>
</tr>
<tr>
<td>Alpha</td>
<td>0.05</td>
</tr>
<tr>
<td>Beta</td>
<td>0.05</td>
</tr>
<tr>
<td>Minimum Volume Per Hull</td>
<td>0.0001</td>
</tr>
<tr>
<td>Plane Downsampling</td>
<td>4</td>
</tr>
<tr>
<td>Hull Downsampling</td>
<td>4</td>
</tr>
<tr>
<td>Enable PCA</td>
<td></td>
</tr>
<tr>
<td>Project Hull Vertices</td>
<td></td>
</tr>
</tbody>
</table>

**Maximum Hulls**

Specify the maximum number of hulls to generate. Valid values range from a minimum of 1 to a maximum value of 1024.

**Maximum Vertices Per Hull**

Defines the maximum number of triangles per convex hull. Valid values range from a minimum of 4 to a maximum value of 1024.

**Concavity**

Specify the maximum concavity of each approximate convex hull. Valid values range from a minimum of 0 to a maximum value of 1.
Resolution

Maximum number of voxels generated during the voxelization stage. Valid values range from a minimum of 10000 to a maximum value of 6400000.

Mode

Select voxel-based approximate convex decomposition or tetrahedron-based approximate convex decomposition.

Alpha

Specify the bias toward clipping along symmetry planes. Valid values range from a minimum of 0 to a maximum value of 1.

Beta

Specify the bias toward clipping along revolution axes. Valid values range from a minimum of 0 to a maximum value of 1.

Minimum Volume Per Hull

Specify the adaptive sampling of the generated convex hulls. Valid values range from a minimum of 0 to a maximum value of 0.01.

Plane Downsampling

Specify the granularity of the search for the best clipping plane. Valid values range from a minimum of 1 to a maximum value of 16.

Hull Downsampling

Specify the precision of the convex hull generation process during the clipping plane selection stage. Valid values range from a minimum of 1 to a maximum value of 16.

Enable PCA

When enabled, the mesh is normalized before applying the convex decomposition.

Project Hull Vertices

When enabled, the output convex hull vertices are projected onto the original source mesh to increase the floating point accuracy of the results.

PhysX modifiers

Modifiers can be added to a PhysX mesh group by choosing the Add Modifier button, and selecting a modifier from the list.

Comment modifier

Add a comment to the file. You can add a comment about changes made to the .fbx file for tracking purposes or notes on export options, for example. Comments don't affect how files are processed and multiple comment modifiers can be added to a mesh group.
**Origin modifier**

![Origin modifier interface](image)

Change the position (translation), orientation (rotation), and scale of a mesh relative to how it was authored.

**Relative Origin Node**

Select the transform relative to which the mesh is processed. By default, the mesh origin is placed at the scene position 0, 0, 0 in the .fbx file.

**Translation**

Sets the position offset of the processed mesh.

**Rotation**

Sets the orientation offset of the processed mesh in degrees.

**Scale**

Sets the scale offset of the processed mesh.

**FBX Settings mesh export**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Meshes contain geometry that doesn't have a skeleton and skinning data. These assets are also called static Meshes. Static meshes are most often used as props and environment objects, but can be used as interactive objects and even player avatars. Entities containing static meshes can be animated through scripts (kinematics) or PhysX simulation (dynamics).

**Contents**

- Export a mesh (p. 436)
- Mesh export video tutorial (p. 437)
- Mesh export tips (p. 437)

**Export a mesh**

1. Copy your .fbx file to a sub directory in your project located at `lumberyard_version\dev\your_project`.
Alternatively, choose Import from the File menu in Lumberyard Editor to copy or move the .fbx file into your project.

2. Locate the .fbx in Asset Browser.

3. Right click the .fbx in Asset Browser and choose Edit Settings to open FBX Settings.

4. In FBX Settings, in the Meshes tab, choose the Hierarchy button to the right of Select meshes and ensure only the meshes required for the Mesh group are selected in the mesh list.

   You can choose Add another mesh to create additional mesh groups. A runtime .cfg asset will be produced for each mesh group. A mesh group can contain any number of meshes from the Select meshes list.

5. Set the properties for each mesh group and add modifiers to each mesh group as required. For information on Mesh group properties and modifiers, see FBX Settings Meshes tab (p. 413).

6. Optional - Set up PhysX colliders for your meshes. For more information, see FBX Settings PhysX export (p. 439).

7. Choose the Update button at the bottom right of the FBX Settings window. A File progress window opens to display information about asset processing. Choose OK to close the File progress window. This step creates or updates the .assetinfo file. Asset Processor automatically processes the .fbx file and generates the runtime .cfg files for meshes, .pxmesh files for PhysX, and .mtl files for materials.

8. Close the FBX Settings window.

Note
When you close FBX Settings after making changes, you might see a pop-up window warning of unsaved changes. This is a known issue and can be disregarded.

Mesh export video tutorial

To learn about exporting static meshes to Lumberyard, watch the following video tutorial.

Exporting static meshes to Lumberyard – Static Meshes

Mesh export tips

- Ensure that each object that needs to perform runtime collision has a PhysX mesh. Low-resolution PhysX meshes work better than high-resolution meshes. Primitives such as a cube, sphere, or capsule are best for optimal physics performance.
- The maximum number of vertices for any static geometry is 65,536. You can export a scene where the total number of vertices exceeds 65,536, but each static geometry piece can't exceed 65,536.

If the combined mesh has more than 65,536 vertices, make the following changes in the FBX Settings tool:

1. In Lumberyard Editor, in the Asset Browser, right-click the .fbx file and choose Edit Settings.
2. In the FBX Settings tool, on the Meshes tab, click Add Modifier and then choose Mesh (Advanced).
3. Clear the Merge Meshes setting. This prevents Asset Processor from merging the meshes, which allows Asset Processor to process the geometry.

FBX Settings actor export

This feature is in preview release and is subject to change.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Actors** contain geometry that has a skeleton and skinning data. An **Actor** is not necessarily a character. Assets that use a skeleton and skinning data to drive animation such as weapons, machines, and foliage, are **Actors**. **Actors** can be animated through keyframe animation imported as **Motions** or through PhysX simulation and other dynamic solvers such as **Touch Bending**.

**Export an actor**

1. Copy your `.fbx` file to a sub directory in your project located at `lumberyard_version\dev\your_project`.

   Alternatively, choose **Import** from the **File** menu in Lumberyard Editor to copy or move the `.fbx` file into your project.

2. Locate the `.fbx` in **Asset Browser**.

3. Right click the `.fbx` in **Asset Browser** and choose **Edit Settings** to open **FBX Settings**.

4. In **FBX Settings**, in the **Actors** tab, ensure the desired root bone of the actor’s skeleton is selected in the **Select root bone** property.

   An **Actor group** may contain only one skeleton hierarchy, and will only export bones that are part of the selected hierarchy.

5. In **FBX Settings**, in the **Actors** tab, choose the **Hierarchy** button to the right of **Select base meshes** and ensure only the meshes required for the **Actor group** are selected in the mesh list.

   You can choose **Add another actor** to create additional actor groups. A runtime `.actor` asset will be produced for each actor group. An actor group can contain any number of meshes from the **Select meshes** list and a single skeleton hierarchy.

6. Set the properties for each actor group. Add modifiers to each actor group as required. For information on **Actor group** properties and modifiers, see **FBX Settings Actors tab** (p. 419).

7. **Optional** - Set up PhysX colliders for your meshes. For more information, see **FBX Settings PhysX export** (p. 439)

8. Choose the **Update** button at the bottom right of the **FBX Settings** window. A **File progress** window opens to display information about asset processing. Choose **OK** to close the **File progress** window. This step creates or updates the `.assetinfo` file. **Asset Processor** automatically processes the `.fbx` file and generates the runtime `.actor` files, `.pxmesh` files for PhysX, and `.mtl` files for materials.

9. Close the **FBX Settings** window.

   **Note**
   When you close **FBX Settings** after making changes, you might see a pop-up window warning of unsaved changes. This is a known issue and can be disregarded.

---

**FBX Settings motion export**

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
**Motions** contain keyframe animated sequences that can be sequenced and blended in **Animation Editor** and applied to actors to create behaviors. Motions must use identical skeleton hierarchies as their actor counterparts.

**Export a motion**

1. Copy your `.fbx` file to a sub directory in your project located at `lumberyard_version\dev\your_project`.
   Alternatively, choose **Import** from the **File** menu in Lumberyard Editor to copy or move the `.fbx` file into your project.
2. Locate the `.fbx` in **Asset Browser**.
3. Right click the `.fbx` in **Asset Browser** and choose **Edit Settings** to open **FBX Settings**.
4. In **FBX Settings**, in the **Motions** tab, ensure the desired root bone of the actor’s skeleton is selected in the **Select root bone** property.
   A Motion may contain only one skeleton hierarchy, and will only export bones that are part of the selected heirarchy.
   A Motion exports a single animation sequence. If the `.fbx` contains multiple animation sequences, additional motions can be added by choosing **Add another motion**.
5. Set the properties for each motion. Add modifiers to each motion as required. For information on Motion properties and modifiers, see FBX Settings Motions tab (p. 426).
6. Choose the **Update** button at the bottom right of the **FBX Settings** window. A **File progress** window opens to display information about asset processing. Choose **OK** to close the **File progress** window.
   This step creates or updates the `.assetinfo` file. **Asset Processor** automatically processes the `.fbx` file and generates the runtime `.motion` files.
7. Close the **FBX Settings** window.
   **Note**
   When you close **FBX Settings** after making changes, you might see a pop-up window warning of unsaved changes. This is a known issue and can be disregarded.

**FBX Settings PhysX export**

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

PhysX colliders are required by **Actors** and **Meshes** to perform collision detection, raycast hit detection, overlap testing, and PhysX simulation. It's possible to use simple parametric primitive shapes in the **PhysX collider** component to create PhysX colliders, but many situations require colliders that more closely resemble the render mesh than simple shapes.

The **PhysX** tab in **FBX Settings** has many options to generate PhysX colliders that offer the best resolution and performance for any use scenario.

**Export PhysX colliders**

1. Copy your `.fbx` file to a sub directory in your project located at `lumberyard_version\dev\your_project`. The `.fbx` file can contain a mesh or an actor.
Alternatively, choose **Import** from the **File** menu in Lumberyard Editor to copy or move the `.fbx` file into your project.

2. Locate the `.fbx` in **Asset Browser**.

3. Right click the `.fbx` in **Asset Browser** and choose **Edit Settings** to open **FBX Settings**.

4. In **FBX Settings**, in the **PhysX** tab, choose the **Hierarchy** button to the right of **Select meshes** and ensure only the meshes required for the **PhysX group** are selected in the mesh list.

The selected meshes can be the visible meshes, or meshes that have been created in a content creation application as physics colliders.

You can choose **Add another physxmesh** to create additional PhysX mesh groups. At least one runtime `.pxmesh` asset is produced for each mesh group. A PhysX mesh group can contain any number of meshes from the **Select meshes** list.

5. Set the properties for each PhysX mesh group, Add modifiers to each PhysX mesh group as required.

Use the **Export as** property to generate the appropriate type of PhysX colliders for your use scenario. **Export as** has three options and one property that are important to understand:

**Triangle mesh**

Triangle mesh generates a triangle mesh collider from the selected meshes. Triangle mesh colliders can offer high precision collisions because they can accurately represent the shape of the visible render mesh. Because triangle meshes can be complex and non-convex, they might carry a performance penalty compared to convex and primitive colliders. Triangle mesh colliders can only be used with static and kinematic entities.

**Primitive**

Primitive either uses the selected primitive shape or determines the most appropriate primitive shape for the collider based on the selected meshes. The primitive shape is automatically fit over the selected meshes. Primitive shapes include box, sphere, and capsule. Primitive shapes offer the highest simulation performance, but might not closely represent the shape of the visible render mesh. Primitives can be used for static, kinematic, and, dynamic entities.

**Convex**

Convex generates a convex hull that encompasses the selected meshes. A convex collider can more accurately represent the shape of the visible render mesh than a primitive collider, at a slight performance cost. Convex colliders can be used for static, kinematic, and dynamic entities.

**Decompose Meshes**

When generating **Primitive** or **Convex** colliders, you have the option to decompose meshes. When enabled, **Decompose Meshes** breaks the selected meshes down into approximately convex parts. Primitives or convex hulls are generated and fit to each decomposed mesh part. Many small colliders that more accurately resemble the shape of the visible render mesh are generated. This option is useful for automatically generating colliders for characters and other complex meshes without authoring colliders in a content creation application or fitting primitive colliders manually.

For further information on **PhysX mesh group** properties and modifiers, see **FBX Settings PhysX tab** (p. 429).

6. Choose the **Update** button at the bottom right of the **FBX Settings** window. A **File progress** window opens to display information about issues encountered during asset processing. Choose **OK** to close the **File progress** window. This step creates or updates the `.assetinfo` file. **Asset Processor** automatically processes the `.fbx` file and generates the runtime `.cfg` files for meshes, `.pxmesh` files for PhysX, and `.mtl` files for materials.
7. Close the **FBX Settings** window.

**Note**
When you close **FBX Settings** after making changes, you might see a pop-up window warning of unsaved changes. This is a known issue and can be disregarded.

### Multiple UV sets for meshes and actors

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can use **FBX Settings** to import multiple UV sets. With multiple UV sets, you can apply a detail or blend layer map to your geometry using UV sets that are independent of the diffuse, normal, and spec UV sets. With multiple UV sets, you can also apply an animated glow that is independent of other texture maps on a mesh.

---

**Topics**
- Processing multiple UV sets (p. 442)
Multiple UV sets

• Materials and multiple UV sets (p. 442)
• Independent tiling and UV sets (p. 442)

Processing multiple UV sets

Lumberyard supports two UV sets per mesh. Asset Processor follows these rules when processing UV sets:

• If the .fbx file contains one or two UV sets, the UV sets are automatically exported.
• If the .fbx file contains three or more UV sets, the first two sets are automatically exported, and the remaining sets are ignored.

Materials and multiple UV sets

By default, FBX Settings creates materials for your exported mesh. You must use the Shader Generation Parameters in the Material Editor to specify how the material should use the second UV set.

Follow these steps to apply the second UV set:
1. In Lumberyard Editor, choose Tools and then Material Editor.
2. In the left pane, navigate to and select the material to use.
3. In the right pane, under Shader Generation Params, do the following to apply the second UV set to one of these features:
   • Blend Layer – Select Blendlayer and Use UV set 2 for blendlayer maps to apply the second UV set to the Second Diffuse Map, Specular, Height, Bump, and Blending Map texture slots.
   • Detail Map – Select Detail mapping and Use UV set 2 for detail map to apply the second UV channel to the Detail texture slot.
   • Emittance Map – Select Use UV set 2 for emittance map to apply the second UV channel to the Emittance and Decal texture slots.

Independent tiling and UV sets

On the Second Diffuse Map, you can set independent values for the blend layer's texture inputs (tiling, rotation, and oscillation). Changes to these values don't affect the tiling, rotation, and oscillation values on the first Diffuse Map. In addition, these values aren't applied to the Blend Map. You can use the Blend Mask Tiling parameter under Shader Params to tile the Blend Map.

You can also set independent values for tiling, rotation, and oscillation for the Detail and Emittance map features.

Follow these steps to set independent values for the texture inputs:
1. In Lumberyard Editor, choose Tools, Material Editor.
2. In the left pane, navigate to and select the material to use.
3. In the right pane, do the following:
   • Under Texture Maps, modify the values for Tiling, Rotator, and Oscillator for your diffuse maps.
   • Under Shader Params, modify the Blend Mask Tiling parameter for your blend map.
FBX soft naming conventions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use soft naming conventions when authoring assets in your content creation tools, such as Autodesk 3ds Max or Maya. Soft naming conventions are prefixes or suffixes that you add to either nodes in your scene or the .fbx file name. In Lumberyard, Asset Processor recognizes these soft naming conventions and then applies an action based on the specified soft naming convention.

Lumberyard provides soft naming conventions as a convenience for content creators to automate steps that are typically done manually in FBX Settings. Depending on the soft naming conventions that you specify, Asset Processor automatically adds those modifiers to the scene settings.

Contents
- Viewing soft naming conventions (p. 444)
- Soft naming convention parameters (p. 446)
- Examples (p. 446)
- Configuring soft naming conventions (p. 447)
- Moving FBX files between projects (p. 448)

Note
The Scene Processing gem is required to use FBX Settings. This gem is enabled by default.

The following table lists current soft naming conventions and the actions that are applied.

Default soft naming conventions

<table>
<thead>
<tr>
<th>Soft Naming Convention</th>
<th>Method</th>
<th>Supported Modules</th>
<th>Example</th>
<th>Asset Processor Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>_phys</td>
<td>Apply suffix to node name.</td>
<td>CryPhysics, PhysX</td>
<td>legMesh_phys</td>
<td>Nodes with the _phys suffix are treated as a physics proxy. A physics proxy modifier is automatically added.</td>
</tr>
<tr>
<td>_lod1</td>
<td>Apply suffix to node name.</td>
<td>Graphics</td>
<td>jack_lod1</td>
<td>Nodes are treated as LoD meshes. The scene settings automatically add an LoD modifier with each suffixed node.</td>
</tr>
<tr>
<td>_lod2</td>
<td></td>
<td></td>
<td>jack_lod2</td>
<td></td>
</tr>
<tr>
<td>_lod3</td>
<td></td>
<td></td>
<td>jack_lod3</td>
<td></td>
</tr>
<tr>
<td>_lod4</td>
<td></td>
<td></td>
<td>jack_lod4</td>
<td></td>
</tr>
<tr>
<td>_lod5</td>
<td></td>
<td></td>
<td>jack_lod5</td>
<td></td>
</tr>
<tr>
<td>_ignore</td>
<td>Apply suffix to node name.</td>
<td>Graphics and CryPhysics</td>
<td>jackSkeleton_ignore</td>
<td>Asset Processor ignores and doesn't process</td>
</tr>
</tbody>
</table>
### Soft Naming Convention

<table>
<thead>
<tr>
<th>Soft Naming Convention</th>
<th>Method</th>
<th>Supported Modules</th>
<th>Example</th>
<th>Asset Processor Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>_anim</td>
<td>Apply suffix to file name.</td>
<td>CryAnimation</td>
<td>jackMoves_anim.fbx</td>
<td>For .fbx file names with the _anim suffix, Asset Processor processes only the animations in the .fbx file.</td>
</tr>
</tbody>
</table>

### Viewing soft naming conventions

You can use the System Entity Editor to view the current soft naming conventions for your project.

**To view the soft naming conventions**

1. In Project Configurator choose Advanced Editor Settings for your project.
2. Expand the Scene Processing Config section to find the soft naming conventions.

**Note**

If the Scene Processing Config section doesn't appear, choose Add Component and then choose Scene Processing Config.
Naming conventions

- Node name setting
  - Pattern: _lod1
  - Matcher: PostFix
  - Virtual Type: LODMesh1
  - Include child nodes

- Node name setting
  - Pattern: _lod2
  - Matcher: PostFix
  - Virtual Type: LODMesh2

- Node name setting
  - Pattern: _lod3
  - Matcher: PostFix
  - Virtual Type: LODMesh3
  - Include child nodes

- Node name setting
  - Pattern: _lod4
  - Matcher: PostFix
  - Virtual Type: LODMesh4
  - Include child nodes

- Node name setting
  - Pattern: _lod5
  - Matcher: PostFix
  - Virtual Type: LODMesh5
  - Include child nodes

- Node name setting
  - Pattern: _phys
  - Matcher: PostFix
  - Virtual Type: PhysicsMesh
  - Include child nodes

- Node name setting
  - Pattern: _ignore
  - Matcher: PostFix
  - Virtual Type: Ignore
  - Include child nodes

- File name setting
  - Pattern: _ignore
  - Matcher: PostFix
  - Virtual Type: Ignore
  - Include child nodes

Select required graph types
Soft naming convention parameters

You can use the System Entity Editor to match the files that use the specified pattern for the node name or file name. The soft naming conventions are matched based on the pattern of the node name or file name that you specify.

There are three approaches available for matching:

**Postfix**

Checks if the node or file name ends in the specified pattern. For file name, the extension isn't included.

**Prefix**

Checks if the node or file name starts with the specified pattern. This approach doesn't include any paths that are in front of the node or file name.

**Regex**

For complex pattern matching, you can use regular expressions to match the entire node (for example, `root.group_a.group_b.mesh`) or file path (for example, `example/file/object.fbx`). This approach is more complex but useful if you want more granular control. For example, you can use regular expression to have all files in an `/animation/` directory export only the animation files.

For **Node name settings**, you can specify the following:

**Virtual type**

Provides the list of virtual types that the nodes are converted to after their pattern matches. You can assign multiple virtual types to a single node.

**Include child node**

If enabled, the virtual type applies to the node and all of its child nodes. You can use this option to apply the change to a parent node and all of its children without adding a separate rule for each child node.

For **File name settings**, you can specify the following:

**Virtual type**

Provides the list of virtual types that the files are converted to after their pattern matches.

**Graph type**

Select one or more graph types that the virtual type is applied to.

**Inclusive**

If enabled, the virtual type applies to all graph node types that are selected. If disabled, the virtual type applies to all graph node types that are unselected.

**Examples**

Using these parameters, you can customize how Asset Processor processes .fbx files that you import.

**Example Node name setting**

You can pair a node name in a source asset file to a predefined Lumberyard virtual type.
1. For **Node name setting**, for **Pattern**, specify _lod1_.
2. For **Matcher**, specify **Postfix**.
3. For **Virtual type**, specify LODMESH1 and select the **Include child nodes** check box.
4. When Asset Processor processes the .fbx file, it checks for node names with the _lod1 suffix. If any nodes match, Asset Processor assigns them the LODMESH1 virtual type, and the change applies to all child nodes.

**Example File name setting**

You can pair a file name in a source asset file to a predefined Lumberyard virtual type.

1. For **Node name setting**, for **Pattern**, specify anim_.
2. For **Matcher**, specify **Prefix**.
3. For **Virtual type**, specify **Ignore**.
4. For **Graph type**, specify IAnimData and select the **Inclusive** check box.
5. When Asset Processor processes your files, it checks for file names with the anim_ prefix and that use the IAnimData graph type. If any file names match, Asset Processor assigns them the **Ignore** virtual type.

**Configuring soft naming conventions**

The preconfigured soft naming conventions might not apply to your workflow. You can change the existing soft naming conventions or create and delete them as needed.

**To change soft naming conventions**

1. In the System Entity Editor, for **Node name settings**, you can specify the following options:
   a. For **Pattern**, specify a value. The matcher uses this value to check against the file.
   b. For **Matcher**, select one of the following:
      - PreFix
      - PostFix
      - Regex
   c. For **Virtual type**, select one of the following:
      - Ignore
      - LODMesh1 – LODMesh5
      - PhysicsMesh
   d. For **Include child nodes**, specify whether the virtual type applies to the node and its children.
2. For **File name setting**, you can specify the following options:
   a. For **Pattern**, enter a value. The matcher uses this value to check against the file.
   b. For **Matcher**, select one of the following:
      - PreFix
      - PostFix
      - Regex
   c. For **Virtual type**, select one of the following:
      - Ignore
      - LODMesh1 – LODMesh5
- **PhysicsMesh**
  - **Graph type**: select the graph types that you want.
  - **Inclusive**: specify whether the virtual type applies to graph node types that you selected in the previous step. If disabled, the virtual type applies to graphs that are unselected.

3. To add a configuration, click the + icon.
4. For **Classes**, choose **File name setting** or **Node name setting** and then click **Save**.
5. To delete a configuration, click the x icon.
6. Click **Save**.

### Moving FBX files between projects

Soft naming conventions are project-specific. As a result, different projects can process the `.fbx` file differently.

**To process the FBX file the same way between projects**

1. In the **FBX Settings** tool, open the `.fbx` file to move.
2. Click **Update** at least once. This creates a manifest (.fbx.assetinfo) file.
3. When you move the `.fbx` file to another project, move the manifest file as well. This ensures that the `.fbx` file is processed the same way, regardless of the specified soft naming conventions.

### Process custom assets with Python Asset Builder

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With Python Asset Builder, you can create Python scripts that process custom assets produced from content creation tools such as Maya and Houdini, or any content tool with a known file format.

To use Python Asset Builder you must enable the Python Asset Builder gem (p. 1192).

### Python Asset Builder terms

The input and output files for a Python Asset Builder are both assets of some type. This documentation uses the following terms to distinguish between the input and output of **Python Asset Builder**.

**Source asset file**

An input asset file, such as an asset produced by a content creation tool, that will be processed by **Asset Processor** to generate a **product asset file**, source dependencies, and build dependencies.

**Product asset file**

An output asset file produced by Python Asset Builder that can be consumed by a game launcher or Lumberyard Editor.

### Writing a Python Asset Builder

There are four steps to create a Python Asset Builder:
• **Create or modify a bootstrap script (p. 449)** - Add a new Python Asset Builder script to a bootstrap location.

• **Register a Python Asset Builder (p. 449)** - Add logic to the Python Asset Builder that registers an asset builder pattern and handlers for job creation and processing.

• **Create jobs with Python Asset Builder (p. 452)** - Define logic in the callback method for `CreateJobs` that responds with the job description to process source asset files.

• **Process job with Python Asset Builder (p. 456)** - Define logic for `ProcessJob` to generate product asset files and dependencies.

---

### Create or modify a bootstrap script

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To make your Python Asset Builder script available to the asset processing system, you must add a `bootstrap.py` file to the path or modify an existing `bootstrap.py` file. We recommend you use a location that is relative to where your Python Asset Builder scripts will be stored, such as one of the following:

- `lumberyard_version\dev\MyProject\Editor\Scripts\bootstrap.py`
- `lumberyard_version\dev\Gems\MyGem\Editor\Scripts\bootstrap.py`

To add your Python Asset Builder, import it in the `bootstrap.py` file.

```python
import asset_builder_my_asset_type
```

The above example assumes that the Python Asset Builder is named `asset_builder_my_asset_type.py`, and that it's in the same directory as the `bootstrap.py` file.

### Register a Python Asset Builder

A Python Asset Builder script must register a file pattern and unique builder ID for the asset builder. To describe the asset builder, use `azlmbr.asset.builder.AssetBuilderDesc`. To bind the script to the asset building process, use `azlmbr.asset.builder.PythonAssetBuilderRequestBus`. You can register the asset builder by a file extension pattern such as `*.myasset` or by a folder and file regular expression pattern such as `^[a-zA-Z]:\MyAssets[\S|\S]*\S*.*$).

When the Python Asset Builder is successfully registered, a handler for the builder ID is used to handle asset builder events to create jobs and process jobs. The script creates a `azlmbr.asset.builder.PythonBuilderNotificationBusHandler` in the module to add the callback handlers.

---

**Contents**

- **PythonAssetBuilderRequestBus (p. 450)**
Register

- AssetBuilderPattern (p. 450)
- AssetBuilderDesc (p. 450)
- PythonBuilderNotificationBusHandler (p. 451)
- Example: Register an asset builder (p. 451)

PythonAssetBuilderRequestBus

The PythonAssetBuilderRequestBus is a singleton EBus that serves methods to enable Python asset builders.

```python
azlmbr.asset.builder.PythonAssetBuilderRequestBus
```

# Registers an asset builder using a builder descriptor.
# input: azlmbr.asset.builder.AssetBuilderDesc
# output: Outcome_bool
'RegisterAssetBuilder'

# Fetches the current executable folder.
# input: N/A
# output: Outcome_string
'GetExecutableFolder'

AssetBuilderPattern

The AssetBuilderPattern is a structure that defines the type of pattern to use to watch for source asset files.

```python
class azlmbr.asset.builder.AssetBuilderPattern
```

- type (azlmbr.asset.builder.AssetBuilderPattern Type) How to use the pattern to watch source assets
- pattern (str) The file path pattern to use

AssetBuilderPattern Type

The pattern type can be either Wildcard or Regex. This informs the asset builder registration how to use the pattern field.

```python
# The pattern is a file wildcard pattern (glob).
azlmbr.asset.builder.AssetBuilderPattern_Wildcard

# The pattern is a regular expression pattern.
azlmbr.asset.builder.AssetBuilderPattern_Regex
```

AssetBuilderDesc

AssetBuilderDesc describes the asset builder for the Asset Processor so it can invoke callbacks for the create jobs event and the process job events for the patterns being that you are registering with the source asset watching system.
The `name` field is a human readable name that is used to trace how the product asset was processed.

The `version` field should be incremented if all the source assets for this builder should be reprocessed, for example, when the product asset contents are redefined.

The `busId` should be in the form of a `azlmbr.math.Uuid`, which can be created once via Python or other methods. You can use Visual Studio's GUID tool, or create one with Python 3:

Create a UUID with Python 3

```python
>>> import uuid
>>> uuid.uuid4()
UUID('639f403e-1b7e-4cfe-a250-90e6767247cb')
```

```python
azlmbr.asset.builder.AssetBuilderDesc
```

```python
class azlmbr.asset.builder.AssetBuilderDesc
- `busId (azlmbr.math.Uuid)`: The builder unique ID
- `name (str)`: The name of the Builder
- `patterns (list of AssetBuilderPattern)`: The collection of file patterns that the builder will use to determine if a file will be processed by that builder
- `version (number)`: Changing this version number will cause all your assets to be re-submitted to the builder for job creation and rebuilding
```

```python
PythonBuilderNotificationBusHandler
```

The notification bus handler is used by Python Asset Builder to call back to the Python script to handle asset building events such as `create jobs` and `process job` to process source asset files. The handler must be created in the global module scope so that the callbacks can stay active.

```python
azlmbr.asset.builder.PythonBuilderNotificationBusHandler
```

```
# Callback function type for creating jobs from job requests.
# input: tuple(azlmbr.asset.builder.CreateJobsRequest)
# output: azlmbr.asset.builder.CreateJobsResponse
'OnCreateJobsRequest'

# Callback function type for processing jobs from process job requests.
# input: azlmbr.asset.builder.ProcessJobRequest
# output: azlmbr.asset.builder.ProcessJobResponse
'OnProcessJobRequest'
```

Example: Register an asset builder

This example code shows how to register an asset builder. The `on_create_jobs` and `on_process_job` functions are stubbed out. See the subsequent topics for information on creating and processing jobs.

```python
import azlmbr.asset
import azlmbr.asset.builder
import azlmbr.bus
```
import azlmbr.math

busId = azlmbr.math.Uuid_CreateString('{CF5C74D1-9ED4-4851-85B1-9B15090DBEC7}', 0)

def on_create_jobs(args):
    # TODO: create jobs logic.
    return azlmbr.asset.builder.CreateJobsResponse()

def on_process_job(args):
    # TODO: process job logic.
    return azlmbr.asset.builder.ProcessJobResponse()

# register asset builder
def register_asset_builder():
    assetPattern = azlmbr.asset.builder.AssetBuilderPattern()
    assetPattern.pattern = '*.newasset'
    assetPattern.type = azlmbr.asset.builder.AssetBuilderPattern_Wildcard

    builderDescriptor = azlmbr.asset.builder.AssetBuilderDesc()
    builderDescriptor.name = "New Asset"
    builderDescriptor.patterns = [assetPattern]
    builderDescriptor.busId = busId
    builderDescriptor.version = 1

    if outcome.IsSuccess():
        # created the asset builder to hook into the notification bus
        h = azlmbr.asset.builder.PythonBuilderNotificationBusHandler()
        h.connect(busId)
        h.add_callback('OnCreateJobsRequest', on_create_jobs)
        h.add_callback('OnProcessJobRequest', on_process_job)
        return h

    # the module global asset builder handler
handler = register_asset_builder()

Create jobs with Python Asset Builder

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you create a job with the Python Asset Builder, the callback for OnCreateJobsRequest will be called with a CreateJobsRequest in a tuple. The callback will return a CreateJobsResponse as the response using the data inside the request.

Contents
- CreateJobsRequest (p. 452)
- CreateJobsResponse (p. 453)
- Example: Create jobs (p. 456)

CreateJobsRequest

CreateJobsRequest provides data for operations related to the enabled platforms. This data is used to build the output JobDescriptor for a specific enabled platform. The CreateJobsRequest data contains input job data that is sent by Asset Processor to the builder for creating jobs.
Create Jobs

**azlmbr.asset.builder.CreateJobsRequest**

class azlmbr.asset.builder.AssetBuilderDesc
- **builderId** (azlmbr.math.Uuid) The builder id to identify which builder will process this
  job request
- **watchFolder** (str) Contains the sub-folder that the source file came from, out of all the
  folders being watched by the asset processor
- **sourceFile** (str) The source file path that is relative to the watch folder (watchFolder)
- **sourceFileUUID** (azlmbr.math.Uuid) The source file's UUID; will be used for its Asset ID
- **enabledPlatforms** (list of azlmbr.asset.builder.PlatformInfo) Information about each
  platform the builder is expected to build

**PlatformInfo**

This structure indicates the platform that has been enabled for the project.

**azlmbr.asset.builder.PlatformInfo**

class azlmbr.asset.builder.PlatformInfo
- **identifier** (str) The ID of the platform such as 'pc' or 'ios'
- **tags** (set of string) The tags such as "console" or "tools" on that platform

**CreateJobsResponse**

The response from the callback determines what work to process for the source asset file. In most cases,
the builder creates a job descriptor for each source asset and for each enabled platform.

**azlmbr.asset.builder.CreateJobsResponse**

class azlmbr.asset.builder.CreateJobsResponse
- **result** (azlmbr.asset.builder.CreateJobsResponse Return Code) The result code from the
  create jobs request
- **sourceFileDependencyList** (list of SourceFileDependency) This is required for source files
  that want to declare dependencies on other source files
- **createJobOutputs** (list of JobDescriptor) JobDescriptor is used by the builder to store
  job related information

**CreateJobsResultCode**

These are the possible result codes from CreateJobs request.

**azlmbr.asset.builder.CreateJobsResponse Return Code**

# Jobs failed to be created.
azlmbr.asset.builder.CreateJobsResponse_ResultFailed

# The builder is in the process of shutting down.
azlmbr.asset.builder.CreateJobsResponse_ResultShuttingDown

# Jobs were created successfully.
azlmbr.asset.builder.CreateJobsResponse_ResultSuccess
SourceFileDependency

This structure defines source asset file dependency information that the builder will send to Asset Processor.

The sourceFileDependencyPath field can be either be a relative path from the assets folder, or an absolute path.

The sourceFileDependencyUUID field is the source asset file UUID part of the asset ID, without the sub-id.

Important
The builder does not need to provide both the sourceFileDependencyUUID and the sourceFileDependencyPath info to Asset Processor. Either one will be sufficient.

azlmbr.asset.builder.SourceFileDependency
class azlmbr.asset.builder.SourceFileDependency
- sourceFileDependencyPath (str) Filepath on which the source file depends
- sourceFileDependencyUUID (azlmbr.math.Uuid) UUID of the file on which the source file depends
- sourceDependencyType (azlmbr.asset.builder.SourceFileDependency Type) Defaults to azlmbr.asset.builder.SourceFileDependency_Absolute

SourceFileDependency Type

azlmbr.asset.builder.SourceFileDependency Type

# Source file depends on other source file
azlmbr.asset.builder.SourceFileDependency_Absolute

# Allow wildcard matches using LIKE
azlmbr.asset.builder.SourceFileDependency_Wildcards

JobDescriptor

JobDescriptor is used by the builder to store job-related information.

The priority field is the value for the jobs within the job queue. A priority value less than 0 means the job's priority is not considered. A priority value of 0 or greater prioritizes the job by value. The higher the value, the higher priority.

Note
Priorities for critical and non-critical jobs are set separately.

The checkExclusiveLock field is a flag to determine whether Asset Processor needs to check the source asset file for exclusive lock before processing the job. Asset Processor will lock and unlock the source asset file to ensure it is not opened by another process. This prevents premature processing of some source asset files that are opened for writing, but have zero bytes for longer than the modification threshold. This will time out if the Asset Processor cannot get an exclusive lock.

The checkServer field determines whether Asset Processor needs to check the server for the outputs of this job before starting to process the job locally. If Asset Processor is running in server mode, then this is used to determine whether it needs to store the outputs of this job on the server.

If the failOnError field is set to True, then all reported errors, asserts, and exceptions cause the job to fail, even if the result code is ProcessJobResult_Success.
Create Jobs

The `setPlatformIdentifier` and `getPlatformIdentifier` methods set and retrieve the platform identifier such as `pc` or `android` for the job description. It is the identifier of the platform from the `PlatformInfo` struct.

```python
class azlmbr.asset.builder.JobDescriptor
- jobParameters (JobParameterMap) Any builder specific parameters to pass to the Process Job Request
- additionalFingerprintInfo (str) Any additional info that should be taken into account during fingerprinting for this job
- jobKey (str) Job specific key, e.g. TIFF Job
- priority (int) Priority value for the jobs within the job queue
- checkExclusiveLock (bool) Attempt to get an exclusive lock file for before we process the job
- checkServer (bool) Check the server for the outputs of this job before we start processing the job
- jobDependencyList (list of azlmbr.asset.builder.JobDependency) This is required for jobs that want to declare job dependency on other jobs
- failOnError (bool) Reported errors, asserts, and exceptions will automatically cause the job to fail
+ set_platform_identifier(platformIdentifier:str) -> None; Sets platform identifier
+ get_platform_identifier() -> platformIdentifier:str; Gets platform identifier
```

**JobParameterMap**

The `JobParameterMap` is a Dictionary of Numbers to Strings. This is a map data structure that holds parameters that are passed into a job for ProcessJob requests. These parameters can optionally be set during the create job function of the builder so that they are passed along to the ProcessJobFunction. The values (key and value) are arbitrary and it is up to the builder on how to use them.

**Example:**

```python
jobParameterMap = {1 : "MyValue", 2 : "Another Value"}
```

**JobDependency**

Job dependency information that the builder sends to the Asset Processor.

```python
class azlmbr.asset.builder.JobDependency
- sourceFile (azlmbr.asset.builder.SourceFileDependency) Source file dependency information that the builder will send to the asset processor
- jobKey (str) JobKey of the dependent job
- platformIdentifier (str) Platform Identifier of the dependent job
- type (azlmbr.asset.builder.JobDependency Type) Type of Job Dependency (order or fingerprint)
```

**JobDependency Type**

```python
azlmbr.asset.builder.JobDependency Type
```
Process job

# This implies that the dependent job should get processed by Asset Processor, if the fingerprint of job it depends on changes.
azlmbr.asset.builder.JobDependency_Fingerprint

# This implies that the dependent job should only run after the job it depends on is processed by Asset Processor.
azlmbr.asset.builder.JobDependency_Order

# This is similar to Order where the dependent job should only run after all the jobs it depends on are processed by the Asset Processor.
# The difference is that here only those dependent jobs matter that have never been processed by Asset Processor.
# Also important to note is the fingerprint of the dependent jobs do not alter the fingerprint of the job.
azlmbr.asset.builder.JobDependency_OrderOnce

Example: Create jobs

This is a simple example of how the asset builder might create jobs when Asset Processor detects a new or changed source asset file in the watch folders of the registered pattern.

```python
# Creates a single job to compile for each platform
def on_create_jobs(args):
    # get the request from the 'args'
    request = args[0]

    # Create job descriptor for each platform
    jobDescriptorList = []
    for platformInfo in request.enabledPlatforms:
        jobDesc = azlmbr.asset.builder.JobDescriptor()
        jobDesc.jobKey = 'My New Asset Job'
        jobDesc.priority = 1
        jobDesc.set_platform_identifier(platformInfo.identifier)
        jobDescriptorList.append(jobDesc)

    response = azlmbr.asset.builder.CreateJobsResponse()
    response.result = azlmbr.asset.builder.CreateJobsResponse_ResultSuccess
    response.createJobOutputs = jobDescriptorList
    return response
```

Process job with Python Asset Builder

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Asset Processor calls the registered callback when it has a job for the builder to begin processing. The callback processes the source asset file, performs all the work inside the temporary directory, creates at least one product asset file, registers the product asset files via a JobProduct entry inside a ProcessJobResponse instance, and returns a success value inside the ProcessJobResponse.

Contents
- ProcessJobRequest (p. 457)
- ProcessJobResponse (p. 457)
- ProcessJobResponse ResultCode (p. 458)
ProcessJobRequest

This is the input into the OnProcessJobRequest function to perform the steps that are needed to process a source asset file. This contains the input data that's needed to process the job for the source asset file.

The jobDescription field contains the job parameters that the OnCreateJobs step created for this job.

The tempDirPath field is the path to the temporary directory that the builder uses to write out intermediate and final product asset files output by this job. The output files are stored inside the temporary directory relative to where the final product asset file will be placed in the cache and pak files.

The sourceFileUUID field is both the unique ID of the source asset file, and the first part of the Asset ID that is used for all the product files output by this job. The only way to differentiate the product files is the subId used in the JobProduct structure.

azlmbr.asset.builder.ProcessJobRequest

class azlmbr.asset.builder.ProcessJobRequest
- sourceFile (str) Relative source file name
- watchFolder (str) Watch folder for this source file
- fullPath (str) Full source file name
- builderGuid (azlmbr.math.Uuid) Builder id
- jobDescription (azlmbr.asset.builder.JobDescriptor) Job descriptor for this job created in OnCreateJobs
- tempDirPath (str) Temp directory that the builder should use to create job outputs for this job request
- platformInfo (azlmbr.asset.builder.PlatformInfo) The information about the platform that this job was emitted for
- sourceFileDependencyList (list of azlmbr.asset.builder.SourceFileDependency) Source file dependency information
- sourceFileUUID (azlmbr.math.Uuid) The UUID of the source file
- jobId (number) Job id for this job, this is also the address for the JobCancelListener

ProcessJobResponse

This is the class that the OnProcessJobRequest callback returns to describe the job's results. The ProcessJobResponse contains job data that indicates the outputs from the job in the outputProducts field, the result code, and schedule sources to be reprocessed.


The outputProducts field indicates what product files need to be copied to the project cache folder. If this is empty, it indicates that the job failed.

The sourcesToReprocess field triggers a rebuild of source assets (via absolute paths) due to the work performed in this job. To reprocess these sources, the builder updates fingerprints in CreateJobs of those builders that process them, like changing source dependencies.
azlmbr.asset.builder.ProcessJobResponse

class azlmbr.asset.builder.ProcessJobResponse:
    - resultCode (azlmbr.asset.builder.ProcessJobResponse ResultCode) baz
    - outputProducts (list of azlmbr.asset.builder.JobProduct) List of job product files
    - requiresSubIdGeneration (bool) Used to determine if legacy RC products need sub ids generated for them
    - sourcesToReprocess (list of str) List of absolute source paths to trigger rebuilds

ProcessJobResponse ResultCode

The process job response result code has both a 'Success' and a 'Failure', but it also has some specific failure cases, such as detecting a crash or network issues.


# When everything was processed correctly the job should return ProcessJobResponse_Success
azlmbr.asset.builder.ProcessJobResponse_Success

# If the job did not create ALL of the expected outputs it should return ProcessJobResponse_Failed
azlmbr.asset.builder.ProcessJobResponse_Failed

# If a tool or internal API returned an exception it should return ProcessJobResponse_Crashed
azlmbr.asset.builder.ProcessJobResponse_Crashed

# If the job detected a cancellation during processing it should return ProcessJobResponse_Cancelled
azlmbr.asset.builder.ProcessJobResponse_Cancelled

# If the job could not reach a remote service or resource it should return ProcessJobResponse_NetworkIssue
azlmbr.asset.builder.ProcessJobResponse_NetworkIssue

JobProduct

A successful processing of a source asset returns one or more JobProduct entries in the ProcessJobResponse's outputProducts field.

The productSubID field is a stable and unique product identifier for each product file created by this process job for the source asset file. It can be any unsigned 32-bit integer that disambiguates different outputs from the same source. If builder source asset files produce only one product, the builder can use 0.

The productAssetType field maps to a C++ AZ::Data::AssetData type ID.

One way to determine the asset type ID from Python is to call the AssetCatalogRequestBus using the asset's display name:

```python
assetType = azlmbr.asset.AssetCatalogRequestBus(azlmbr.bus.Broadcast, 'GetAssetTypeByDisplayName', "Font")
print(f'Asset type {assetType}')
```
dependenciesHandled indicates to the **Asset Processor** that the builder has output all possible dependencies for this source asset file for this job product file. This can be true if there are no output product files. This should be set to True only if the builder outputs its dependencies or the output product doesn’t have dependencies. When set to False, **Asset Processor** emits a warning that dependencies have not been handled.

The **JobProduct** constructor takes in a product file name (relative to the source asset path), an asset type (Uuid), and a product sub-ID number.

```python
class azlmbr.asset.builder.JobProduct
    - productFileName (str) Relative or absolute product file path
    - productAssetType (azlmbr.math.Uuid) The asset type ID this product file loads into the asset catalog
    - productSubID (number) A stable product identifier
    - productDependencies (list of ProductDependency) Product assets this source asset depends on
    - pathDependencies (set of ProductPathDependency) Specifies dependencies by relative path to source assets
    - dependenciesHandled (bool) Indicates that the builder has output all possible dependencies
+ JobProduct(productFileName:str, productAssetType:azlmbr.math.Uuid, productSubID:number) A constructor to set a job product
```

**ProductDependency**

The product dependency information that the builder sends to **Asset Processor** to indicate that a product asset depends on another product asset during load.

```python
class azlmbr.asset.builder.ProductDependency
    - dependencyId (azlmbr.math.Uuid) Asset ID of this product asset dependency
```

**ProductPathDependency**

The **ProductPathDependency** represents the product's dependency information that the builder detected on another product file (relative to the source asset path). If the source asset ID can be determined, we recommend that you use the **productDependencies** instead to indicate the product's dependency information in terms of asset IDs. It's preferable to depend on product files whenever possible, to avoid introducing unintended dependencies.

The **dependencyType** field indicates if the path points to a source file or a product file.

```python
class azlmbr.asset.builder.ProductPathDependency
    - dependencyPath (str) Relative path to the asset dependency
    - dependencyType (azlmbr.asset.builder.ProductPathDependency Type) Indicates if the dependency path points to a source file or a product file
```
**ProductPathDependency Type**

ProductPathDependency Type indicates how to use the dependency path in the ProductPathDependency. A dependency on a source file is converted into dependencies on all product files produced from the source. It's preferred to depend on product files whenever possible, to avoid introducing unintended dependencies.

```python
# If the source asset depends on another product asset file the value should be SourceFile
azlmbr.asset.builder.ProductPathDependency_Type_SourceFile

# If the source asset depends on another source asset the value should be ProductFile
azlmbr.asset.builder.ProductPathDependency_Type_ProductFile
```

**Example: Process**

This is a simple example of how the asset builder might process a job when Asset Processor detects a new or changed source asset file in the watch folders of the registered pattern.

```python
# Using the incoming 'request' find the type of job via 'jobKey' to determine what to do
def on_process_job(args):
    try:
        # Get request information
        request = args[0]

        # Prepare output folder
        basePath, _ = os.path.split(request.sourceFile)
        outputPath = os.path.join(request.tempDirPath, basePath)
        os.makedirs(outputPath)

        # Write out a simple file
        productFileNameOnly = 'myfile.txt'
        filename = os.path.join(outputPath, productFileNameOnly)
        file = open(filename, "w+")
        file.write('some data')
        file.close()

        # Prepare output entry data
        productOutputs = []
        basePath, sceneFile = os.path.split(request.sourceFile)
        assetProductName = os.path.join(basePath, productFileNameOnly)
        outputFilename = os.path.join(request.tempDirPath, assetinfoFilename)

        # Create job product entry
        assetType = azlmbr.math.Uuid_CreateString('{F67CC648-EA51-464C-9F5E-4A9CE41A7F86}', 0)
        product = azlmbr.asset.builder.JobProduct(assetProductName, assetType, 0)
        product.dependenciesHandled = True
        productOutputs.append(product)

        # Fill out response object
        response = azlmbr.asset.builder.ProcessJobResponse()
        response.outputProducts = productOutputs
        response.resultCode = azlmbr.asset.builder.ProcessJobResponse_Success
        return response
    except:
        # An exception should record a proper failure
        response = azlmbr.asset.builder.ProcessJobResponse()
```

Version 1.28

460
response.resultCode = azlmbr.asset.builder.ProcessJobResponse_Crashed
return response
Working with component entities

The component entity system provides a modular and intuitive construction of game elements. The component entity system works at both the system level and the entity level. It employs reflection, serialization, messaging using the event bus (EBus), fully cascading prefabs (slices), and the ability to drag-and-drop and edit entities and their components in Lumberyard Editor.

Use the following Lumberyard Editor tools to improve the workflow for the component entity system.

- Entity Outliner (p. 462)
- Entity Inspector (p. 475)

**Note**
The component entity system replaces the (now legacy) Object and Entity System in the Amazon Lumberyard Legacy Reference.

See the following topics to add components to entities and customize their properties in Lumberyard Editor.

For information on programmatically creating your own custom components, see Programmer’s Guide to Entities and Components (p. 973).

**Topics**

- Entity Outliner (p. 462)
- Entity Inspector (p. 475)
- Working with Components (p. 479)
- Working with Slices (p. 510)
- Component Reference (p. 532)
- Converting Entities with the Legacy Converter (p. 963)
- Programmer’s Guide to Entities and Components (p. 973)
- Gameplay Bus (p. 1030)
- Reflecting Lumberyard Classes, Methods, and EBus Interfaces (p. 1032)

**Entity Outliner**

The **Entity Outliner** shows all the entities and slices in the level. You can view the parenting hierarchies, lock selections, show and hide entities in the viewport, and create search filters.
To open the Entity Outliner

1. In Lumberyard Editor, choose Tools, Entity Outliner.
2. In the Entity Outliner, you can create, select, search, and filter for entities.

You can select multiple entities by doing one of the following in the Entity Outliner:

- Use CTRL + Left Mouse Click to select multiple entities one at a time to be part of a multi-selection.
- Use Shift + Left Mouse Click to select a range of entities.

**Note**

- Entities in white are freestanding entities.
- Entities in blue are part of a slice. For more information, see the slice icons (p. 511).

**Working with Entities**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Topics**

- Creating an Entity (p. 463)
- Reordering Entities (p. 464)
- Hiding and Showing Entities (p. 465)
- Locking Entities (p. 465)
- Search and Filter for Entities (p. 466)

**Creating an Entity**

Entities are objects with which the player interacts.

To create an entity
Do one of the following:

- In the Lumberyard Editor viewport, right-click and choose Create entity. This creates an entity at the cursor location with a basic Transform (p. 878) component, which gives the entity a 3D location in the level.
- In the Entity Outliner (p. 462), right-click and choose Create entity. This creates an entity in the center of your viewport.
- Use the Asset Browser (p. 296) to create entities that already have the preferred configuration, depending on the particular asset.

For example, if you drag a .cgf mesh asset from the Asset Browser into the viewport, Lumberyard creates an entity, adds a Mesh (p. 684) component, and assigns the asset to the Mesh component. The same is possible for particles, slices, and Lua scripts.

To assign entities to a parent entity

1. In the viewport or the Entity Outliner, right-click and then choose Create entity. This creates an empty entity, which will be the parent entity.
2. Do one of the following:
   - In the Entity Outliner, select and drag the child entities on top of the new parent entity.
   - Select one or more entities that are to be the child entities. In the Entity Inspector, for the Transform component, beside the Parent Entity property, click the picker icon and then select the parent entity in the viewport.
   - Select one or more entities that are to be the child entities. From the Entity Outliner, drag the parent entity into the Entity Inspector and drop it beneath the Transform component into the Parent Entity property.

Reordering Entities

When you create entities or instantiate slices, they appear at the bottom of the list in the Entity Outliner.

To reorder entities in the Entity Outliner

1. To move an entity, right-click an entity and choose Move up or Move down.
2. You can also drag and drop one or more entities into the preferred order. Select and drag the entity until a white line appears in your preferred location.

3. To make an entity the child of another entity, select and drag the entity to its intended parent. A white box appears around the parent entity.

Hiding and Showing Entities

You can hide entities so that they don't appear in the viewport, so that the viewport shows only the entities that you want. If you hide a parent entity, all children entities are also hidden. You can also hide child entities within a parent entity.

To hide and show entities

1. In the Entity Outliner, click the eye icon next to the entity. The crossed-out eye icon indicates that the entity doesn't appear in the viewport.
2. To show the entity, click the icon again.

Locking Entities

You can lock entities so that they can't be selected in the viewport, so that you can select only the entities that you want to edit. If you lock a parent entity, all children entities are also locked. You can also lock child entities within a parent entity.
To lock entities

1. In the Entity Outliner, click the lock icon next to the entity. The icon appears white when the entity is locked.

2. To unlock the entity, click the lock icon again.

Search and Filter for Entities

For levels that have many entities, you can search and filter for the entities that you want. Enter text in the search filter box to find specific entities.

To search for an entity by name

1. In the Entity Outliner, enter the name for the entity.
2. To filter by component, click the filter icon.
3. Enter component names in the search field that appears or scroll and select the entities that you want. Any entity that has the specified component appears in the results.

Example

The entities that appear have either the Camera or the Trigger Area components attached.
4. To clear search results, click **Clear**.

**To search for an entity by ID**

- In the **Entity Outliner**, enter the complete ID for the entity. Partial matches and wildcard searches are not supported.

You can also sort entities so that they appear in the order that you want in the **Entity Outliner**.

**To sort entities**

1. In the **Entity Outliner**, click the sort icon.

2. Choose the following options:
   - **Sort: Manually** – Manually organize entities. See Reordering Entities (p. 464).
   - **Sort: A to Z** – Sort entities alphabetically, in ascending order.
   - **Sort: Z to A** – Sort entities, in descending order.
   - **Scroll to Selected** – When you select an entity in the viewport, the **Entity Outliner** scrolls to that entity. If you select multiple entities, the **Entity Outliner** scrolls to the last selected entity.
Working with Entities

- **Expand to Selected** – When you select an entity in the viewport, the **Entity Outliner** expands the hierarchy to show any child entities.

You can also select an entity in the **Entity Outliner** to find it in the viewport or in reverse. This feature helps you find the entities that you want, especially in large levels.

**To locate an entity or slice**

1. In the **Entity Outliner**, right-click an entity and choose **Find in viewport**. The viewport navigates to the corresponding entity.

2. In the **Entity Outliner**, right-click the slice or slice entity and choose **Find slice in Asset Browser**. The **Asset Browser** navigates to the corresponding slice.
3. In the viewport, right-click a slice or entity and choose **Find in Entity Outliner**. The **Entity Outliner** navigates to the corresponding item.

![Entity Outliner menu](image)

**Working with Layers**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/beta/3d-engine/) or visit the [AWS Game Tech blog](https://aws.amazon.com/beta/3d-engine/blog/) to learn more.

Use the Lumberyard layer system to organize level data into discrete files. The layer system segments level content so that members of a development team can work on different aspects of a level asynchronously.

Layers are standard Lumberyard entities with a special editor-only layer component. In your game, this component appears as an empty entity in your hierarchy. When you export your game, the behavior you designed is unchanged and all entities in your level are added to the exported data.

**Topics**

- Creating a Layer (p. 470)
- Modifying a Layer (p. 471)
- Adding Entities to a Layer (p. 473)
- Saving a Layer (p. 473)
- Recovering a Layer (p. 474)
- Layer-Specific Components (p. 474)
Creating a Layer

When you create a layer, you can add entities to that layer. This helps you organize the content in your game. For example, you might create a layer for all character entities and another layer for vegetation.

To create a layer

1. In Lumberyard Editor, choose Tools, Entity Outliner.
2. In the Entity Outliner, right-click and choose Create layer.
3. With the layer selected in the Entity Outliner, you can modify its properties in the Entity Inspector.

<table>
<thead>
<tr>
<th>Layer Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Layer name.</td>
</tr>
<tr>
<td></td>
<td><strong>Warning</strong> If you rename a layer, this creates a new layer the next time you save your level.</td>
</tr>
<tr>
<td>Status</td>
<td>You can set the following options:</td>
</tr>
</tbody>
</table>
## Working with Layers

### Layer Properties

<table>
<thead>
<tr>
<th>Layer Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Start active – Layer is active when the level starts.</td>
<td></td>
</tr>
<tr>
<td>• Start inactive – Layer is inactive when the level starts. Overrides the status property of the entities contained in the layer.</td>
<td></td>
</tr>
<tr>
<td>• Editor only – Layer is only active in editor mode. However, entities within that layer that are not marked Editor only will still appear in the game.</td>
<td></td>
</tr>
</tbody>
</table>

### Color

Specifies the display color of the layer. Use the color picker or manually type in the color value field. This is the line color used to show the layer container in the Entity Outliner.

Default: gray

### Save as binary

Saves the layer in binary format. Binary assets, which are not human readable, serialize faster and are smaller on disk.

---

**Modifying a Layer**

After you create a layer, you can modify it by adding entities, reorganizing its hierarchy, adding nested layers, renaming the layer, and so on.

**To show a layer's context menu**

1. In the Entity Outliner, right-click the layer.
2. You can do the following in the context menu.

   Actions highlighted in yellow affect the selected layer. The other options are standard context menu actions that don't affect the selected layer.
The following options in the context menu perform actions on the selected layer.

<table>
<thead>
<tr>
<th>Layer Context Menu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create child entity</td>
<td>Adds an entity to the layer as a direct descendant.</td>
</tr>
<tr>
<td>Save layer</td>
<td>Saves the selected layer.</td>
</tr>
<tr>
<td>Assign to layer</td>
<td>Moves the selection and makes it a child of a new or chosen layer.</td>
</tr>
<tr>
<td>Duplicate</td>
<td>Makes a copy of the layer and its contents.</td>
</tr>
<tr>
<td>Delete</td>
<td>Removes the layer (and contents) from the scene.</td>
</tr>
<tr>
<td>Open pinned Inspector</td>
<td>Opens a copy of the Entity Inspector that is locked to the properties of the layer.</td>
</tr>
<tr>
<td>Rename</td>
<td>Renames the layer.</td>
</tr>
<tr>
<td>Move up and Move down</td>
<td>Moves the selected layer up or down one entry.</td>
</tr>
<tr>
<td>Find in viewport</td>
<td>Frames the viewport camera to see the contents of the layer.</td>
</tr>
</tbody>
</table>

**Layer Hierarchies**

You can nest layers within other layers. This is useful if you want to organize the entities in your level. This behavior is similar to creating hierarchies for parent and child entities.
Note
You can't make a layer a child of a non-layer entity and you can't save a layer in a slice.

You can nest layers to break up your level into smaller, more workable sections. If you are creating a large level, for example, you might have a single vegetation layer. If you have just one vegetation layer, then only one environment artist could edit this layer at a time. To allow multiple artists to work on the vegetation layer at once, you can nest other layers within the vegetation layer and assign each nested layer to different artists. This helps build a well-organized hierarchy to keep the game's structure efficient.

Adding Entities to a Layer

Layers can contain freestanding (non-slice) entities and slices.

To add an entity to a layer

- In the Entity Outliner, do one of the following:
  - Select and drag an entity to a layer or within the layer hierarchy.
  - Right-click an entity, pause on Assign to layer, and then select a layer.

Saving a Layer

The component entity system saves references to layers and their hierarchies in the level data. When you add or remove a layer from your level, you must save your level before making more changes. If you don't save your level, layers and their contents will not load correctly the next time you open the level.

Lumberyard layers are saved as .layer files in the level_name/layers directory. The layer's filename is saved as layer_name.layer. If a layer is nested within another layer, then the parent layer name is prepended to the layer filename.

When a layer contains unsaved changes, an asterisk (*) appears next to the layer name. After you save the level or the layer, the asterisk is removed.

Layer names at the same hierarchy level must be unique. Layers at the same hierarchy level with duplicate names display a warning (!) and can't be saved until you rename them.
To save your level and all layers
- In Lumberyard Editor, choose File, Save or press Ctrl+S.

To save specific layers only
1. Select the layer you want to save, or press CTRL and then select multiple layers.
2. In the Entity Outliner, right-click the selection and choose Save.

Recovering a Layer
If you delete a layer from a level in Lumberyard Editor, you can reimport it.

To reimport a deleted layer
1. Using a file browser, copy onto your desktop the layer file for the layer that you want to recover, such as level_name\layer\layer_name.layer.
2. In Lumberyard Editor, create a new layer in your level and enter the same name as the deleted layer.
3. Save the level and close Lumberyard Editor.
4. Copy the layer file from your desktop into Lumberyard's layer directory, such as level_name\layers.
5. Rename the copied layer file to match and replace the layer that you created in Lumberyard Editor.
6. Reopen the level. The newly created layer now references the recovered layer information.

Layer-Specific Components
A layer is simply an entity with special rules. As such, you can add layer-specific components to layers. By default, Lumberyard doesn't contain any layer-specific components, but you can create your own, such as special layer components for streaming or tags.

Any given component can appear in only one context menu. By default, Lumberyard has the Game, System, and Layer contexts for components.

You can test creating a layer-specific component by editing the Comment component.

To modify the Comment component
1. In a text editor, open the EditorCommentComponent.cpp file.
2. Change the AZ_CRC attribute to Layer and delete the CRC value.
3. Save the file.
4. In Lumberyard Editor, add the Comment component to a layer.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Entity Inspector** manages all the components for each entity. Select an entity in the **Entity Outliner** or the viewport to see the attached components in the **Entity Inspector**.

**To open the Entity Inspector**

1. In Lumberyard Editor, choose **Tools, Entity Inspector**.
2. Select an entity in the viewport or the **Entity Outliner**.
3. In the **Entity Inspector**, you can see the following:
   - **Name** – Name of the entity. You can enter a different name for the entity.
   - **Entity Icon** – Customizable icon to help you recognize entities in the viewport.
   - **Status** – Active status of the entity. When the level starts, the entity can be active, inactive, or active but only in editor mode.
   - **Entity ID** – If this entity ID is called out in messages, errors, or asserts, you can find the entity by searching for it in the **Entity Outliner**.
   - Components attached to the entity appear below.

Use the **Entity Inspector** to do the following:

- Add components to entities (p. 479)
- Modify component properties (p. 481)
- Remove, copy, cut, and paste components (p. 480)
- Set the entity status (p. 476)
- Customize the entity icon (p. 477)
- Pin an entity's inspector (p. 478)
- Create your own help topic for your custom component (p. 482)
Note
For a list and descriptions of available components, see Component Reference (p. 532). You can also click the Help icon in the header of each component to open a help topic.

Setting an Entity Status

By default, an entity starts as active in a level. When you create a game, you can specify that an entity remain inactive until activated through some mechanism such as a script or player action. You can also set an entity as editor only if you want to disable an entity during gameplay mode or you want to create test entities or visual comments for users working in your game.

To set an entity’s status

1. In the Entity Outliner or the viewport, select an entity.
2. In the Entity Inspector, choose the Status drop-down menu, and select one of the following options:
   - Start active – Entity is active when the level starts.
   - Start inactive – Entity is inactive when the level starts.
   - Editor only – Entity is only active in editor mode.
3. When you set an entity as Start Inactive or Editor only, select the entity to view its status in the Entity Outliner and the viewport.

Example Start Inactive

Inactive entities have a strikethrough icon and inactive text appears in the viewport.
Customizing the Entity Icon

The default icon for an entity without any added components is the Transform ( ▶️ ) component's icon. When you add another component, the icon changes to the first component that you add to that entity.

You can also specify your own icon.

To customize an entity icon

1. In the Entity Inspector, click the icon image at the top.
2. Choose Set custom icon.
3. Select an icon from your game project directory.
Pinning an Entity's Inspector

You can pin an entity's inspector to keep it open and visible even when you select another entity. You can pin inspectors for multiple entities, and also pin multiple inspector instances of the same entity. This helps you compare the entities and their components to each other.

A pinned inspector has the following features:

- Always shows the pinned entity even when you select a different entity.
- Functions like the main Entity Inspector window.
- Closes when you open a different level or exit Lumberyard.
- If you convert a loose entity to a slice, the pinned inspector points to the new slice entity that corresponds to the previously loose entity.
- Persists when entering and exiting game mode.
- Updates all pinned inspectors for a particular entity when you modify that entity.

You can pin an inspector from the Entity Outliner or the Entity Inspector.

To pin an inspector

1. Select an entity.
2. Do one of the following:
   a. In the Entity Outliner, right-click the entity and then choose Open pinned Inspector.
   b. In the Entity Inspector, click the pin icon.
3. In Lumberyard Editor, you can view the pinned entity inspectors.

**Example**

[Image of pinned entity inspectors]

---

**Working with Components**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the component entity system tools to create entities and add components to them. Before you can work with components, you must create an entity (p. 463).

**Topics**

- Adding Components to an Entity (p. 479)
- Managing Components (p. 480)
- Editing Component Properties (p. 481)
- Disabling and Enabling Components on Entities (p. 482)
- Editing Components in the Viewport (p. 483)
- Using Manipulators (p. 490)
- Working with the Viewport Interaction Model (p. 494)

**Adding Components to an Entity**

After you create an entity, you can add components to it.
Managing Components

To add a component to an entity

1. In the Entity Inspector, click Add Component or right-click and choose Add Component.
2. Components are grouped by category, such as Animation, Camera, and Gameplay. To find a specific component quickly, enter the name into the search bar at the top of the component list.
   
   **Note**
   You can pause on the component name to see a description of that component.
3. Select the component.
4. Specify your settings for the component.
5. If you add a component that requires another component to function, the Entity Inspector displays the following message.

   For example, the Ragdoll component also requires the Skinned Mesh component. Click Add Required Component and select the required component.

   **Note**
   Components must have the required dependencies to appear in game mode. The Entity Inspector displays a warning message for the following scenarios:
   
   - Incompatibilities exist between components
   - A component is missing a required component
   - A component is a duplicate

   You must resolve the issue before you can use the component for the entity. Otherwise, the component is disabled.

Managing Components

You can also use the context menu in the Entity Inspector to remove, copy, and reorder components.

To manage components from the context menu

1. In the Entity Outliner or in the viewport, select the entity.
2. In the Entity Inspector (p. 475), right-click the component, or click the menu button in the header, and then select one of the following:
   
   - **Add component** – Add a component to the entity
   - **Delete component** – Remove a component from an entity
   - **Cut component** – Cut a component from one entity and paste it onto a different entity
   - **Copy component** – Copy a component and then paste it onto the same entity, which creates a duplicate component that you can customize
   - **Paste component** – Copy a component and then paste it onto a different entity
   - **Enable component** – Make a component available
   - **Disable component** – Make a component unavailable
• **Move component up** – Move a component up the list
• **Move component down** – Move a component down the list
• **Move component to top** – Move a component to the top of the list
• **Move component to bottom** – Move component to the bottom of the list

You can also click and drag individual components into a desired order. Organize the components for an entity in any order that makes sense to you. The custom order for your components is saved to the level. The component order has no significance to its function.

**Note**
- The Transform (p. 878) component is automatically added to an entity and can't be moved from the component list.
- Some actions are disabled, depending on the context. For example, you can't paste a component if you haven't copied one.

## Editing Component Properties

You can edit properties for your components such as changing the mesh asset file for the Mesh component.

**To edit component properties**

1. In the **Entity Outliner** or the viewport, select the entity.
2. In the **Entity Inspector**, select the component and edit its properties.
3. To undo a change that you made to a property, press Ctrl+Z. To redo the change, press Ctrl+Shift+Z.
4. To edit multiple entities at the same time, select the entities that you want and make your changes. The changes that you make to the first entity propagate to all selected entities.

**Note**
- If your entity is part of a slice instance, any properties that you modify from the source slice asset appear in orange in the **Entity Inspector**.
- To save a change to a slice, see Modifying a Slice and Saving Changes (p. 519)
- To revert an override, see Reverting and Forcing Overrides (p. 522)

For more information about slices, see Working with Slices (p. 510).

## Copying and Pasting Asset References

In the **Entity Inspector**, you can copy an asset reference from one component and paste it into another component. You can copy assets as scripts, mesh assets, particle effect libraries, cubemap assets, and so on.

**To copy and paste an asset reference**

1. In the **Entity Inspector**, select the component and right-click the asset reference field.
2. Choose **Copy asset reference**.
3. Select another component, right-click the asset reference field, and choose **Paste asset reference**.
Creating Custom Component Help Topics

If you have created your own components, you can point the component header's help icon to your own documentation.

To do this, add the `HelpPageURL` attribute to your component reflection.

For example:

```csharp
Attribute(AZ::Edit::Attributes::HelpPageURL, "https://docs.aws.amazon.com/lumberyard/latest/userguide/component-comment.html")
```

Disabling and Enabling Components on Entities

After you add components to entities, you can choose to disable and enable them as needed. This can help you see how different components interact with the entity. For example, if you create an entity and add the `Area Light`, `Environment Probe`, and `Point Light` components to it, you can see how the different light components interact with the entity.

You can disable the components that you don't want and reenable them later. Disabled components are read only and don't activate, generate warnings, or export with game data.

**To disable and enable components**

1. In the **Entity Outliner**, select the entity that you want.
2. In the **Entity Inspector**, under **Add Component**, right-click the component that is associated with the entity.
3. Choose **Disable component** to disable the component.
Disabled components are dimmed with stripes.

4. To reenable a disabled component, right-click the component and choose **Enable component**.

**Editing Components in the Viewport**

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you're working with components, you can lock a single component for editing. You can't edit any other components that are attached to the entity. This feature prevents you from accidentally editing other components that are attached to the entity. For example, if you have an entity with attached
Spline and Mesh components, you can lock the Spline component, make your changes to the spline, and stop editing.

This feature supports the following components:

- Box Shape (p. 818)
- OccluderArea (p. 701)
- Portal (p. 717)
- Polygon Prism Shape (p. 708)
- Spline (p. 866)
- Tube Shape (p. 894)
- VisArea (p. 918)

Topics

- Editing a Single Component in the Viewport (p. 484)
- Editing Multiple Components in the Viewport (p. 487)

If you want to add this feature to other components, you must make code changes to Lumberyard Editor. For more information, see the Programmer's Guide to Component Mode (p. 992).

Editing a Single Component in the Viewport

For components that support this feature, the component is locked when you edit its properties in the viewport.

To edit a single component in Component Mode

1. In Lumberyard Editor, create an entity. See Creating an Entity (p. 463).
2. In the Entity Inspector, choose Add Component and select the Spline component.

Example

An entity with the Spline component appears in the viewport.
3. In the viewport, double-click the spline or, in the Entity Inspector, click Edit.

**Example**

The manipulator appears in the viewport. The vertices turn red. When you select a vertex, the manipulator appears.
All other components attached to the entity are dimmed in the **Entity Inspector**, and you can't edit them.

4. Select a vertex and make changes such as adding, moving, or deleting vertices.
5. When you've finished, press **Esc** or, in the **Entity Inspector**, on the component, click **Done**.

The vertices become yellow again.

**Editing Multiple Components in the Viewport**

In some cases, Component Mode supports editing multiple components. This is useful when you want to lock a specific component type for editing. Component Mode supports this feature in the following cases:
• An entity has dependent components, such as the **Tube Shape** and **Spline** components
• You select multiple entities that all have the same component that supports Component Mode (such as a **Box Shape**)

**To edit multiple components in Component Mode**

1. In Lumberyard Editor, create an entity. See *Creating an Entity* (p. 463).
2. In the **Entity Inspector**, choose **Add Component** and then select components that have this feature. See *supported components* (p. 484).

1. In the **Entity Inspector**, choose **Edit**. You can't edit the **Transform** component, but the **Spline** and **Tube Shape** components are active.
2. Make your changes to the component.
3. To switch between components, do one of the following:
   • Select the entity and press **Tab**. The **Entity Inspector** highlights the selected component.
   • In the Lumberyard Editor toolbar menu, choose **Edit** and choose **Edit Next** or **Edit Previous**.

**Example**

The **Transform** component is dimmed, but you can edit the **Spline** and **Tube Shape** components.
In the following example, a radius for the Tube Shape is selected.
Note

The Lumberyard Editor Edit menu shows the available actions for the selected component. You can specify different shortcuts except for the Save, Undo, and Redo actions.

Example

For example, if you select a vertex point for a Tube Shape component, the Edit menu shows the available shortcuts.

Using Manipulators

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
With manipulators, you can edit certain component properties directly in the viewport instead of the component's property window. Manipulators help you visualize and achieve the results that you want for your components. For example, you can add, move, and delete vertices on a Spline (p. 866) component to create the exact curvature that you want for your level.

By default, some components support manipulators. When you author new components, you can also manually add manipulator support. For more information, see Editing Components in the Viewport (p. 483).

Topics
- General Case Manipulators (p. 491)
- Special Case Manipulators (p. 491)
- Advanced Use Case Manipulators (p. 491)
- Manipulator Appearance (p. 492)
- Creating New Manipulators (p. 492)
- EBus Request Bus Interface (p. 493)

General Case Manipulators

The following are general case manipulators.

**Linear Manipulator**
- Moves the position of an object along one axis.

**Planar Manipulator**
- Moves the position of an object along a plane, which is two axes, such as XY, YZ, or XZ.

**Surface Manipulator**
- Moves the position of the object over the terrain.

**Translation Manipulator**
- Aggregate manipulator that combines multiple linear and planar manipulators (depending on which axes are defined) and an optional surface manipulator.

**Selection Manipulator**
- Performs no movement but provides an interface to detect when part of a component has been clicked.

Special Case Manipulators

The following special case manipulators appear only on applicable components.

**Line Segment Selection Manipulator**
- Determines the location on a line segment over which the mouse is hovering to insert a new point.

**Spline Selection Manipulator**
- Determines the location on a spline over which the mouse is hovering to insert a new point.

Advanced Use Case Manipulators

You can provide a streamlined editing experience by creating a system that manages manipulators across components with similar requirements.
For example, you can edit the vertex positions for the following components in the viewport.

- OccluderArea (p. 701)
- Polygon Prism Shape (p. 708)
- Portal (p. 717)
- Spline (p. 866)
- VisArea (p. 918)

EditorVertexSelection is an example of a component that encapsulates manipulator behavior that can be shared by multiple components – in this case, all components that require vertex editing.

The following core classes are related to EditorVertexSelection.

- HoverSelection (Spline and LineSegment variants)
- VertexContainerInterface
- VariableVertices
- FixedVertices (for when selection and editing of vertices is allowed, but vertices cannot be added or removed)
- Many other manipulators such as Translation, LineSegmentSelection, and SplineSelection.

**Manipulator Appearance**

A manipulator's visual appearance does not determine its behavior. This means that any manipulator can have any view. This is abstracted through the ManipulatorView class. Lumberyard provides a number of concrete views, such as Line, Cylinder, Cone, Box, Quad, and Billboard.

You can choose which view to use with a particular manipulator. All manipulators delegate their rendering to a ManipulatorView. The view also updates the bounds of the manipulator to handle interaction and picking. ManipulatorView provides several utility functions to make creating views easier. You can ignore these and create custom implementations if preferred.

**Creating New Manipulators**

To create a manipulator, you must derive from BaseManipulator. By doing so, you can observe how existing manipulators are implemented and follow a similar approach.

The following are key functions to implement:

- OnMouseDown
- OnMouseUp
- OnMouseOver
- Draw
- Invalidate
- SetBoundsDirty

To call the mouse functions, in the derived manipulator constructor, Attach must be called; for example, AttachLeftMouseDownImpl();.

The new manipulator should provide callbacks in the overridden mouse functions so that classes can handle their own unique use cases.
**Note**
Adding new manipulators is often unnecessary. The existing set provides most of the behavior that is required. For example, you can use the manipulator `LinearManipulators` in a variety of scenarios. This is recommended for most users.

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information, see [Working with the Event Bus (EBus) system](p. 1851).

**Manipulator Manager Requests**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RegisterManipulator</code></td>
<td>Clears the active manipulator and destroys what is associated with it (for example, &quot;Remove Vertex&quot;).</td>
<td><code>BaseManipulator&amp; None</code></td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td><code>UnregisterManipulator</code></td>
<td>Clears the active manipulator and destroys what is associated with it (for example, &quot;Remove Vertex&quot;).</td>
<td><code>BaseManipulator&amp; None</code></td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td><code>SetActiveManipulator</code></td>
<td>Override what the currently bound active manipulator is set, or set to null.</td>
<td><code>BaseManipulator* None</code></td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td><code>DeleteManipulatorBound</code></td>
<td>Delete the specified manipulator bound.</td>
<td><code>RegisteredBoundId None</code></td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td><code>SetBoundDirty</code></td>
<td>Mark the bound of the manipulator dirty, so that it's excluded from mouse hit direction. Bound will usually be recalculated the next time it is drawn.</td>
<td><code>RegisteredBoundId None</code></td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td><code>SetAllBoundsDirty</code></td>
<td>Mark bounds of all manipulators as dirty.</td>
<td>None</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td><code>UpdateBound</code></td>
<td>Update the bound for a manipulator,</td>
<td><code>ManipulatorId</code></td>
<td><code>RegisteredBoundId</code></td>
<td>No</td>
</tr>
</tbody>
</table>
Working with the Viewport Interaction Model

This feature is experimental and is missing some core functionality.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The new Viewport Interaction Model replaces the old interaction model in the viewport. This feature combines free and local selection so that you can more easily select, move, and modify entities in the viewport. With the Viewport Interaction Model, you can create custom reference spaces to move your entities. With custom reference spaces, you can move your selected entities in relation to the reference that you specified. The Viewport Interaction Model also unifies entity and component manipulators so that you can interact with entities and components in the same way. Gizmos are deprecated.

The Viewport Interaction Model provides the following benefits:

- Improves the ability to select your preferred entity, especially for nested hierarchies or levels that have many entities.
- Prevents you from accidentally deselecting an entity. The entity that you select is locked for editing.
- Adds keyboard and mouse actions that you can apply to your selected entities. For example, you can freely switch between local, parent, and world space to make changes to your selected entities.
- Establishes an editing pattern that matches the Component Mode feature. For more information, see Editing Components in the Viewport (p. 483).

See the following concepts for the Viewport Interaction Model:

**Manipulator**

With manipulators, you can edit component properties directly in the viewport instead of the Entity Inspector.

For more information, see Using Manipulators (p. 490).

**Custom Reference Target**

When you select an entity, you can specify whether to move that entity in relation to another entity.

**Viewport Interaction Model Limitations**

This feature has the following limitations:

- The Viewport Interaction Model supports component entities only and doesn't support legacy entities, such as terrain and vegetation.
- When you enable the Viewport Interaction Model, this feature removes some toolbar options in the Lumberyard Editor Interface (p. 183), such as vertex snapping.
- You can't change or customize shortcuts for the Viewport Interaction Model.
Enabling Viewport Interaction Model

By default, the Viewport Interaction Model isn't enabled for the editor. You can enable it from the Global Preferences window.

To enable the Viewport Interaction Model

1. In Lumberyard Editor, choose Edit, Editor Settings, Global Preferences to open the Preferences window.
2. Select Enable New Viewport Interaction Model (EXPERIMENTAL).
3. Restart Lumberyard Editor.
4. Open a level and, in the Asset Browser, select and drag two assets into the viewport.
5. In the viewport, select an entity. A manipulator appears on the entity. The entity is locked for editing. If you click another entity in the viewport, the first entity remains selected.

Use the following shortcuts on a selected entity.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press Space or Esc.</td>
<td>Deselects the entity. You can also double-click elsewhere in the viewport to deselect the entity.</td>
</tr>
<tr>
<td>Press and hold Ctrl, then click and drag the manipulator in the viewport.</td>
<td>Selects the manipulator so that you can move it in the viewport. This enables you to adjust the manipulator independent from the entity.</td>
</tr>
<tr>
<td>Press and hold Ctrl and Alt and click a target entity.</td>
<td>Moves the manipulator to the target entity that you clicked. The target entity becomes a reference space for manipulating the selected entity.</td>
</tr>
</tbody>
</table>

See Using Custom Reference Spaces (p. 498).

Modifying the Transform

When you move the manipulator to another location in the viewport, you can modify the scale, rotation, or translation of the entity. This feature makes it easier for you to manage your entities.

To modify the transform

1. In the viewport, select an entity. The manipulator appears.
2. Click and drag the manipulator to transform the selected entity.
3. Press and hold Ctrl, and click and drag the manipulator to another location.
4. Use the manipulator again to adjust the scale, rotation, or translation of the entity.

**Example**

In the following example, the manipulator is moved away from the car, switches to rotate mode, and rotates the car.

The following are shortcuts to switch between the different manipulator modes.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press 1.</td>
<td>Switches to translation mode.</td>
</tr>
<tr>
<td>Press 2.</td>
<td>Switches to rotate mode.</td>
</tr>
<tr>
<td>Press 3.</td>
<td>Switches to scale mode.</td>
</tr>
<tr>
<td>Press and hold Ctrl, then scroll the middle mouse button.</td>
<td>Switches between translation and rotation modes.</td>
</tr>
<tr>
<td>Press and hold Ctrl, move the mouse to a target location, then press the middle mouse button.</td>
<td>Moves the manipulator to the location that you specified in the viewport.</td>
</tr>
</tbody>
</table>

**Reference Space Model**

You can use the Viewport Interaction Model to create reference spaces for your selected entities. This enables you to customize your selected entities in relation to a reference space.

When working with reference spaces, remember the following rules:

- New selections always default to their parent space. This will be the world coordinate space if the entity does not have a parent.
- Holding Shift aligns the manipulator to the world coordinate space.
• Holding Alt aligns the manipulator to an entity's local coordinate space.
• You can easily access custom spaces by moving the manipulator (press Ctrl and click and drag) or picking a target entity as a reference space (press Ctrl and Alt and click a target entity).

The Viewport Interaction Model simplifies the mental mode for you and doesn't require you to keep track manually of the last space (local or world) that you were using. You can specify a target and define that target as custom reference spaces. This new reference space model covers any arbitrary transformation.

Switching Between Local and World Space

You can use the manipulator to switch between local and world space without losing focus of your entity selection in the viewport.

To switch between local and world space

1. In the viewport, select a child entity that has a parent. Your selection always defaults to parent space.
2. Use the manipulator to modify the entity.
3. To switch to world space, press and hold Shift. To switch to local space, press and hold Alt. Use the manipulator to modify the entity.

Note

• If you selected an entity that doesn't have a parent, your selection defaults to world space.
• Parent space is the transform that the parent entity has in the hierarchy. If an entity doesn't have a parent, the entity uses world space instead.

Example

In the following example, the entity moves in world space because the car does not have a parent. When you select the child entity (the tire), the manipulator switches to the parent space of that entity (the car).
The following are shortcuts to work with the reference space mode.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select an entity or group of entities and press and hold <strong>Shift</strong>.</td>
<td>Switches the manipulator to world space.</td>
</tr>
<tr>
<td>Select a group of entities and press and hold <strong>Alt</strong>.</td>
<td>Switches the manipulator to local space.</td>
</tr>
<tr>
<td>Select an entity or group of entities and press <strong>R</strong>.</td>
<td>Resets the transform to the parent space.</td>
</tr>
<tr>
<td>Select a manipulator and press <strong>Ctrl + R</strong>.</td>
<td>Resets the manipulator back to parent space. This shortcut applies only if the manipulator transform has been modified. <strong>Note</strong> Currently, this shortcut doesn’t affect the manipulator in scale mode.</td>
</tr>
<tr>
<td>Select an entity or group of entities, press and hold <strong>Ctrl</strong> and <strong>Alt</strong>, and press the middle mouse button on a target entity.</td>
<td>Copies the <strong>Transform</strong> settings for the entity that you selected and applies them to a target entity. The <strong>Transform</strong> settings depend on the manipulator mode (translation, rotation, scale) that you’re in. For example, if you select an entity and the manipulator mode is rotation, this shortcut applies the rotation values to a target entity. This ensures that both entities (your selection and the target) have the same rotation.</td>
</tr>
<tr>
<td>Select an entity and press <strong>P</strong>.</td>
<td>Toggles pivot mode. For more information, see Using Pivot Mode (p. 508).</td>
</tr>
</tbody>
</table>

### Using Custom Reference Spaces

You can adjust the manipulator independent of the entity to create a custom reference space. You can also pick another target entity as a reference space. When you specify a reference space, the entities that you select move in relation to that space.

**To create a custom reference space**

1. In the viewport, select an entity.
2. Select a manipulator mode, such as translation.
3. Press and hold **Ctrl** and **Alt** and click a target entity in the viewport. This matches the manipulator to the translation or orientation of the target entity.

**Example**

In the following example, the excavator entity becomes a reference space for the car. When the manipulator moves the car, the car moves in relation to the reference space.
4. Use the manipulator to modify the entity.

The following are shortcuts for working with custom reference spaces.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press and hold Ctrl and drag the manipulator in the viewport.</td>
<td>Moves the manipulator to another reference point.</td>
</tr>
<tr>
<td>Press and hold Ctrl and Alt and click another entity.</td>
<td>Moves the manipulator to a target entity to use as a reference point for the selected entity.</td>
</tr>
</tbody>
</table>

**Note**
The manipulator and the custom reference space are context-based and apply only when you select an entity. If you don't select an entity, there's no manipulator, and you can't create a reference space. The reference space doesn't persist across selections. However, if you move the reference space, you can undo that action.

**Using Group Selections and Reference Spaces**

In the viewport, you can select multiple entities. This is called a group selection. You can use the following shortcuts for group selection.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click and drag in the viewport to create a box around entities.</td>
<td>Selects multiple entities.</td>
</tr>
<tr>
<td>Press and hold Ctrl and click and drag in the viewport.</td>
<td>Deselect multiple entities.</td>
</tr>
<tr>
<td>Press and hold Ctrl and click a target entity.</td>
<td>Adds or removes an entity from the selection. If you already selected a group of entities, this</td>
</tr>
</tbody>
</table>
When you select a group of entities, the Viewport Interaction Model follows the same model as selecting a single entity. For a group selection, the manipulator acts like a temporary parent entity. This means that the feature works the same for a single entity or a group of entities. By default, a manipulator defaults to the common parent of the group selection. If you select entities that don’t share a common parent, world space is used.

**To use group selections and reference spaces**

1. In the viewport, select multiple entities.
2. Press and hold Ctrl and click and drag the rotation manipulator.
3. Press Shift to change from custom to world space.
4. Use the manipulator to modify the entities.

**Example**

In the following example, three entities are selected. Because there is no common parent of the group, the manipulator defaults to world space. The entities now move in relation to the parent (the manipulator).

5. Press and hold Ctrl and Alt and click a target entity. This selects a reference space.
6. Use the manipulator to modify the entities.

**Example**

In the following example, a group of entities move in relation to the reference space.
7. Press and hold **Ctrl**, and move the manipulator in the viewport. This creates a custom reference space.

**Example**

In the following example, you can select a group of entities and move the manipulator in the viewport to create a custom reference space.

8. Select child entities of the same parent entity and use the manipulator to change them. When you select a group of entities that share the same parent, the manipulator defaults its orientation to the parent.
Example

In the following example, two child entities (the tires) are selected. The manipulator rotates the child entities in relation to its parent (the car).

9. Press and hold Alt while modifying a group of selected entities. This enables you to control changes to the entities in local space. You can quickly iterate and modify your entities and view the changes in the viewport.

Example

In the following example, you can press and hold Alt during translation. This moves each entity in their own local space.
10. Select child entities from different parent entities and press and hold Alt. You can modify multiple entities at the same time, even if they have different parents.

**Example**

In the following example, child entities from different parents are selected. The manipulator modifies the child entities at the same time.

11. Press and release Alt to dynamically change the reference space.

**Example**

In the following example, the manipulator changes the scale for the selected entities, switching from world to local space.
Resetting the Transform

You can reset the transform for an entity, so that you return the entity to its previous location.

**To reset the transform**

- In the viewport, select an entity or group of entities and press **R**.

Note that the effect of the reset depends on the current manipulator mode. For example, if you are in the translation mode, the translation is reset for the selected entity or entities.

The reset operation can be modified using the **Shift**, **Alt**, or **Ctrl** keys to perform different kinds of resets. Each of the reset operations can also be found in the *Edit* menu. See the following table for a description of each option.

<table>
<thead>
<tr>
<th>Reset Type</th>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Entity Transform</td>
<td><strong>R</strong></td>
<td>Resets the transform to parent space.</td>
</tr>
<tr>
<td>Reset World Transform</td>
<td><strong>Shift + R</strong></td>
<td>Resets the transform to world space. This shortcut ignores hierarchy depth.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> This has no effect in the translation or scale mode.</td>
</tr>
<tr>
<td>Reset Local Transform</td>
<td><strong>Alt + R</strong></td>
<td>Resets the transform to local space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> This has no effect in the translation mode.</td>
</tr>
<tr>
<td>Reset Manipulator</td>
<td><strong>Ctrl + R</strong></td>
<td>Resets the manipulator back to default parent space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> This shortcut doesn't reset the manipulator scale.</td>
</tr>
</tbody>
</table>

**Example**

In the following example, you can reset the transform as the entity transforms are changing and return them to their original transform.
Matching the Transform

Instead of manually copying values from one entity's position to another, you can use the ditto feature to share an entity's transform data from one entity to another. This feature enables you to duplicate the same transform data for your entities. For example, to make your child entities face the same direction, you can select the entities and use the ditto feature to apply the change to the entities at once.

To match a transform using the ditto feature

1. In the viewport, select an entity.
2. Press and hold Ctrl and press the middle mouse button on a target entity. After the target is selected, the current selection transform matches that of the target.

Example

In the following example, the ditto feature shares an entity's orientation with another entity. Both entities have the same value.
The following is a shortcut to use the ditto feature.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select an entity or group of entities, press and hold Ctrl, and press the middle mouse button on a target entity.</td>
<td>Applies the transform from a target entity to a selection.</td>
</tr>
</tbody>
</table>

### Ditto a Group Selection

You can use the ditto feature for a group of entities. This makes it easier for you to modify multiple entities at once.

**To match the transform for a group of entities**

1. In the viewport, select a group of entities.
2. Press and hold Ctrl and press the middle mouse button to match an entity's transform data to the group manipulator.

**Example**

In the following example, you can select entities (a group selection of tires) and use the ditto feature to match a target entity (the car).
See the following shortcut to use the ditto feature.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select a group of entities, press and hold Ctrl, and press the middle mouse button on a target entity.</td>
<td>Matches the translation, rotation, or scale for the group of entities to that of the target entity.</td>
</tr>
</tbody>
</table>

**Ditto a Group Selection to Local Space**

You can ditto a group of entities to local space so that you can modify entities in local space in relation to another entity.

**To ditto a group selection to local space**

1. In the viewport, select a group of entities.
2. Press and hold Ctrl and Alt and press the middle mouse button to select a target entity. This sets the local space of each entity in the selected group to the target entity that you specified.

**Example**

In the following example, select a group of entities and use the ditto feature to set them to the local space of the target entity.
See the following shortcut to use the ditto feature for a group selection.

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select an entity or group of entities, press and hold <strong>Ctrl</strong> and <strong>Alt</strong>, and press the middle mouse button on a target entity.</td>
<td>Sets the local space of each selected entity to the target entity that you specified.</td>
</tr>
</tbody>
</table>

### Using Pivot Mode

You can use pivot mode to use the default pivot or the entity's center as the manipulator point of reference. This feature is useful if you want to modify multiple entities using a central point of reference.

**Note**
Currently, this is a global toggle and doesn't support mixed use cases, such as using the center of an entity to snap it to the pivot of another entity.

**To use pivot mode**

1. In the viewport, select an entity or group of entities.
2. Press **P** to switch between the pivot and the entity's center.

**Example**

In the following example, the selected entities pivot around the manipulator.
Using the Camera Space

A camera space manipulator is built into the rotation manipulator. You can rotate entities based on the editor’s camera view.

**To use the camera space**

1. In the viewport, select an entity.
2. Adjust the camera around the selected entity in the viewport. You can use the following camera shortcuts to adjust your view.

**Adjusting the Camera View**

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hold right mouse button and move the mouse.</td>
<td>Enables you to look freely in the level.</td>
</tr>
<tr>
<td>Hold Alt + left mouse button and move the mouse.</td>
<td>Orbits the camera around a fixed point.</td>
</tr>
<tr>
<td>Do one of the following while moving the mouse in the viewport:</td>
<td>Dollies the camera to zoom in or out.</td>
</tr>
<tr>
<td>• Hold Alt + right mouse button.</td>
<td></td>
</tr>
<tr>
<td>• Hold middle mouse button + right mouse button.</td>
<td></td>
</tr>
<tr>
<td>Hold middle mouse button and move the mouse.</td>
<td>Pans the camera.</td>
</tr>
<tr>
<td>Hold Alt + middle mouse button and move the mouse.</td>
<td>Pans the camera inverted.</td>
</tr>
</tbody>
</table>
Moving the Camera

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press W, A, S, and D.</td>
<td>Moves the camera forward, left, back, and right.</td>
</tr>
<tr>
<td>Press Q and E.</td>
<td>Moves the camera down and up.</td>
</tr>
<tr>
<td>Press and hold Shift and press W, A, S, and D.</td>
<td>Quickly moves the camera in the specified direction.</td>
</tr>
</tbody>
</table>

Example

In the following example, you can adjust the camera space so that your selected entity remains the center of focus in the viewport.

Working with Slices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A slice is a collection of configured entities (p. 462) that is stored as a single unit in a reusable asset. You can use slices to conveniently group entities and other slices for reuse. Slices are similar to prefabs but are part of the new component entity system. Slices can contain component entities, whereas prefabs cannot. Unlike prefabs, slices can be nested into a fully cascading hierarchy. For example, a level, a house, a car, and an entire world are all slices that depend on (cascade) from a number of other slices.

You can generate a slice asset that contains any number of entities that you have placed and configured. These entities can have arbitrary relationships. For example, they can exist in a parent/child transform hierarchy, although this is not required.
A slice can contain instances of other slices. Modifications of a slice instance within another slice causes the changes to be stored in the instance as overrides (in the form of a data differential or delta). The modifications stored can be changes such as entity additions, entity removals, or component property changes.

When you build a level, you place certain items repeatedly, such as a prop, furniture, or a piece of landscaping. To create multiples of an item, you could copy and paste it throughout your level. Doing this would result in each item having its own independent properties. If you wanted to change one property, such as making all the motorbikes blue instead of red, you would have to modify each one. This is time consuming and inefficient.

With Lumberyard's slices, you can modify one instance of the item and then save that change to all the other instances of your item within the game project. Saving changes to the other instances is not an automatic process. This means that you can modify a slice instance and then not save the changes. This makes that instance unique. You can also detach a slice instance so that it does not inherit saved modifications. That detached slice instance becomes a regular entity with individual properties.

Slices contain entities, including their components and properties, and may also contain instances of other slices. This ability to nest slices without flattening the hierarchy is a feature unique to Lumberyard.

Slices are saved as `.slice` files within your game project directory.

**Example**

In the **Entity Outliner**, you can identify different entities by their icon and color.

![Entity Outliner](image-url)
### Icon Description

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrateWallParent</td>
<td>Entities that are white are standalone. They are not part of a slice instance.</td>
</tr>
<tr>
<td>gsg_maze_floodlight</td>
<td>Entities that are blue are part of a slice instance.</td>
</tr>
<tr>
<td>Maze_PickupObjective GraphicNeutralPos</td>
<td>Entities with lines indicate a hierarchy with a parent and child. In this example, the parent and child entities are part of a slice instance.</td>
</tr>
<tr>
<td>gsg_maze_doorway</td>
<td>Entities that are orange have overrides, which means they have different component property values than the source slice.</td>
</tr>
<tr>
<td>DoorWay_Parent</td>
<td>Parent entities that appear with a dot indicate that a child entity has an override. In this example, the DoorWay_Parent entity has three children, which each have overrides.</td>
</tr>
<tr>
<td>TimerParent</td>
<td>Entities that are shaded are the slice roots. In this example, the TimerParent entity and its grandchildren are slice roots.</td>
</tr>
</tbody>
</table>

### Topics

- Anatomy of a Slice (p. 514)
- Creating a Slice (p. 516)
- Instantiating a Slice (p. 517)
- Creating Nested Slices (p. 517)
- Modifying a Slice and Saving Changes (p. 519)
- Detaching Entities and Slice Instances (p. 522)
- Reverting and Forcing Overrides (p. 522)
- Slice Reloading (p. 524)
- Working with Dynamic Slices (p. 524)
- Slice Favorites (p. 526)
- Converting Slices with the Slice Upgrade Pipeline (p. 529)

The sections in this chapter use the following terminology to describe working with slices.

### Source Slice

The source slice is the `.slice` asset that is saved in the game project directory when you create a slice.
Owning Slice

Entities in a slice instance originate from an owning slice. For example, a tire entity from an instance of `wheel.slice` is owned by `wheel.slice`. Slices automatically pass down changes to any slice instance entities that they own. For example, say that you update the material reference on a tire entity that is owned by a `wheel.slice` instance. When you push that change to the source `wheel.slice`, Lumberyard updates all other tire entities of `wheel.slice`.

Entities can have more than one owning slice. To continue the example, `motorcycle.slice` owns two instances of `wheel.slice`. The tire entity within the two `wheel.slice` instances, which is within `motorcycle.slice`, inherits from both the wheel slice and the motorcycle slice.

Slice Instance

A slice instance is a distinct instantiation of a source slice. The slice instance inherits entities and properties from its source slice (the `.slice` file) and is updated when the source slice changes.

You can modify one slice instance. To apply those modifications to all instances of the source slice, you can save the changes to the source slice asset.

When you work with one instance of a slice, those changes are exclusive to that slice until you push the changes to the source slice.

Slice Entity

Entities that are owned by a slice are called slice entities. A slice entity inherits from its source slice. The Entity Outliner distinguishes slice entities from non-slice entities by color. Slice entities are blue, and non-slice entities are white.

Saving Slice Overrides

Saving slice overrides means to commit entity modifications from an instance to a source slice. When you use the `Save slice overrides` command, Lumberyard prompts you to choose which modifications or overrides to save. For a nested slice, you can also choose which slice in the hierarchy receives the overrides.

After you save overrides, Lumberyard automatically updates all instances of that slice. If any of those slice instances have unsaved overrides or modifications, it does not accept updates to those overrides. You can revert a slice’s entities and properties to resync to the source slice.

For more information, see Modifying a Slice and Saving Changes (p. 519).

Note

Lumberyard does not save the `Translate` value in the `Transform` component to the slice. This value determines an entity’s position in the level and as such is excluded from affecting other slice instances.

Overriding

Changing any part of a slice, such as adding a component or modifying a property, results in an override or modification. This means that the slice instance is different from the source slice. Overrides to component properties are marked in a bold orange font in the Entity Inspector to distinguish it from properties that haven’t changed.

If you save those overrides, they are saved to all the slice instances. This means they are no longer overrides and the font returns to normal.Overrides that you don’t save to the slice are maintained for that instance even when other changes are received from the source slice.

You can also force an override for a property that you haven’t yet modified, as well as revert overrides to an entity, component, or property.

For more information, see Reverting and Forcing Overrides (p. 522).
Detaching

You can detach an entity within a slice. This removes it from that slice instance, and that entity no longer receives updates when changes are saved to the source slice.

You can also detach any slice instance. This means that it no longer receives any changes saved to the slice.

For more information, see Detaching Entities and Slice Instances (p. 522).

Nesting

When a slice instance is nested, that means that it is a child of another slice instance.

If you push a nested slice instance to its parent's source slice, then the nested slice instance inherits from both its original source slice and any source slices that own its parents.

A nested slice instance shares a transform hierarchy with its parent. For more information, see Transform Hierarchy.

Inheriting

Slice instances inherit data from all source slices in their hierarchy. When a slice instance inherits from multiple source slices, inheritance priority occurs from top to bottom in the outliner hierarchy.

A slice instance can lose its inheritance. If you modify a slice instance and then don't save those changes to the source slice, that slice instance loses its inheritance. To restore the inheritance, you must either reset the slice instance or save the changes.

Transform Hierarchy

A transform hierarchy defines movement, rotation, and scaling of entities in both the editor and at runtime. A child entity shares the transform data of its parent. This means that you can move, rotate, or scale a parent, and its children always follow. You can, however, move, rotate, and scale a child independent of its parent.

Slice Hierarchy

A slice hierarchy defines a slice instance's relationship to its source slices. An entity inside of a nested slice can potentially inherit from multiple source slices. When this happens, slice data at the top of the slice hierarchy overrides slice data at the bottom of that hierarchy.

Data Patch

Generically speaking, a data patch captures the difference between two serializable objects, such that when you take the first object and apply the patch, you get the second object as a result. With respect to slices, when you edit an instance of a slice in the Editor and create an override, a data patch is generated and saved to a slice file. A data patch stored in a slice preserves the information needed to modify the source slice to create a slice with overrides. This may be as simple as changing a value in a field, or something more complex such as adding or removing components or entire entities.

Anatomy of a Slice

The following diagram illustrates an example slice A, which contains references to two other slices B and C. Slice A has two instances each of B and C:
Each instance contains a data patch, which may be empty if no changes or overrides are present. If the instantiation of slice B in slice A has been modified in comparison with the source asset B, the data patch contains the differences. When slice A is instantiated again, it contains instances of slice B, but with the modifications applied. Any nonoverridden fields propagate through the hierarchy. If you change a property value in the slice B asset on disk, the instance of B contained in slice A will reflect that change — if the property for that instance has not already been overridden, as reflected in the instance's data patch.

In addition to references to other slices, slices can contain zero or more entities. These entities are original to this slice and are not acquired through referenced slice instances. A slice does not have to contain references to other slices. A slice that contains only original entities (as represented by the bottom box in the diagram) and no references to other slices is called a *leaf slice*.

Slice files in a Lumberyard project have the extension "*.slice". They are stored in an XML format.

**Anatomy of a Slice Data Patch**

When a slice contains a reference to another slice, and an override is made to this reference, the difference between the two is captured in a data patch. The overrides can be simple, such as a change to a value in a field, or more complex, such as the addition or removal of entire components or entities. The format of the serialized data in a slice file depends on the version of Lumberyard.

**Readable XML Data Patch Format (v1.23+)**

Component names, data types, and version information are readily identifiable in this data patch format. There is no need to launch Lumberyard Editor to see this information. The following example shows that:

- This data patch is for the EditorCommentComponent.
Creating a Slice

A slice can contain any number of entities that have their own transform hierarchy. The slice, however, must have a single transform root. This means that you can change the size and position of child entities without affecting the parent entity. But if you change the size or position of the parent entity, the child entities automatically adjust as well.

To create a slice

1. In the viewport or in the Entity Outliner, select one or more entities to include in the slice.
2. In the viewport or the Entity Outliner choose Create slice.
3. Save the slice file to the preferred location. We recommend that you save slices to your game project folder. Name your slices meaningfully and organize them into directories and subdirectories.
**Note**

- If you want Lumberyard Editor to add a number to your slices by default (for example, `NewSlice001`), you can set this option in the General Settings. For more information, see General Settings (p. 239).
- To save multiple entities in a slice, they must have a single transform root. If they don't, then a Cannot Create Slice dialog box appears, giving you the opportunity to fix this issue by creating a root entity. Select Yes to create a root entity while creating the slice.

You also create a slice from another slice. This creates a new slice and that slice will no longer inherit from its previous source slice.

For example, if you have a slice instance named `car.slice` and you want to use that slice instance to create a new slice, you can create a detached slice named `bus.slice`. You will then have two separate slices that no longer inherit from each other.

**To create a slice from an existing slice and remove inheritance**

1. In the Entity Outliner, select and right-click the slice instance that you want.
2. Choose Create detached slice.
3. Enter a file name for the slice and then save.

### Instantiating a Slice

Loading an instance of a slice in your level is called instantiating a slice.

**To instantiate a slice**

- Do one of the following:
  - In the viewport, right-click and choose Instantiate slice. In the Pick Slice dialog box, select a slice, and then click OK.
  - Drag a slice asset from the Asset Browser (p. 296) into the viewport. The slice is instantiated at the drop location.

### Creating Nested Slices

A nested slice contains instances of other slices. Nested slices can store their own overrides for component and entity properties. They can also contain their own entities.

For example, suppose that you have a slice called `wheel.slice`, composed of one wheel. You have another slice called `motorcycle_body.slice`, composed of an engine, seat, and frame. You create a new slice called `motorcycle.slice` and add two instances of the wheel slice and one instance of...
the motorcycle body slice, and then save the changes. The body and wheel slices are now nested in the motorcycle slice.

**To create a nested slice**

1. In the **Entity Outliner**, drag one or more slices onto another slice. The slices become child entities of the slice you dropped them on.
2. Right-click the parent entity and choose **Create slice**.
3. The **Create Slice** dialog box will appear giving you a choice of creating a fresh slice or a nested slice. Choose **Nested Slice**.

![Create Slice dialog box](image)

4. Enter a file name for the slice and then save.

When you create a slice hierarchy, you can select the root slice for that slice. This is useful if you have a large slice hierarchy and you want to quickly navigate between the slice instances.

**To move up or down the slice hierarchy**

1. In the **Entity Outliner**, right-click a child entity in the slice hierarchy and choose **Select slice root**.
2. In the dialog menu, view the slice hierarchy for the selected slice.
3. Choose a slice to select that slice in the **Entity Outliner**.

**Note**

- To select the top slice in the hierarchy, select a slice and press **Shift+R**.
- To select the next parent slice up, press **R**.

**Modifying a Slice and Saving Changes**

If you modify an entity that is part of a slice instance, you create an override for that change. An override means that the slice instance has different component properties than its source slice. You can verify that a slice has overrides with the following:

- In the **Entity Inspector**, the components and the properties that differ from the source slice appear orange. For more information, see **Entity Inspector** (p. 475).
- In the **Entity Outliner**, the entity appears orange and has a dot if a child slice has an override. For more information, see the **Entity Outliner table** (p. 511).

**Example**

In the **Entity Inspector**, for this slice instance, the **Mesh asset** property is different from its source slice, so the component and the property name appears orange.
In the **Entity Outliner**, the slice instances with overrides appear orange.

You can then choose to save that override to all other slice instances.

**To save a slice override**

1. In the viewport or the **Entity Outliner**, right-click the entity, choose **Save slice overrides**, and then select the slice. This saves the override to all other slice instances. The number of changes appears next to the slice.

   **Note**
   You can also right-click the component property in the **Entity Inspector** and choose **Save field override**.

**Example**

This shows a slice instance with eight total changes: one removal and seven updates.

2. To save multiple slice overrides, in the **Entity Outliner** or the viewport, select your entities, right-click and then choose **Save slice overrides, Save slice overrides (Advanced)**.
3. In Save Slice Overrides – Advanced dialog box, for Property, select the changes that you want and then select the target slices to which you want to save the overrides. You can also select or deselect the following check boxes.

- **Changed** – Overrides to existing entities.
- **Added** – Added entities.
- **Removed** – Deleted entities.

4. When finished, choose **Save Selected Overrides**.

**Note**

- To quickly save changes to a selected slice, select the slice and press **Alt+S**.
- Because slices can be nested, you can save modifications to one or more levels of the nested slice hierarchy. To save changes to the top slice root in the hierarchy, select the root slice and press **Ctrl+Alt+S**.
Detaching Entities and Slice Instances

You can detach entities within a slice or detach the entire slice instance.

- When you detach an entity, that entity no longer inherits any changes from its source slice. Use this feature so that the entity becomes a standalone entity.
- When you detach a slice instance, the entities in the slice instance hierarchy no longer inherit changes from the former source slice. Use this feature if you want this slice instance (and all its child slices) to become standalone entities.
- This option appears only when you are managing slice hierarchies for a slice instance.

To detach specific entities within a slice instance

1. In the Entity Outliner, select one or more entities within a slice instance.
2. Right-click the selection and choose Detach, Selection.

   A dialog box appears that tells you that the entity will be converted into a non-slice entity and that the action cannot be undone.
3. Choose Detach. The selected entities are now standalone.

You can also detach the entire slice instance so that all entities in the instance are standalone.

To detach a slice instance

1. In the Entity Outliner, select any part of a slice.
2. Right-click the selection and choose Detach, Instance.

   A dialog box appears that tells you that the detached instance will no longer inherit any changes from its slice and that all entities in the slice instance will be converted into non-slice entities.
3. Choose Detach. All entities are now standalone.

Reverting and Forcing Overrides

When you change any part of a slice instance, such as adding a component, removing a component, or modifying a component property, you create an override. Overrides to component properties are highlighted with bold orange text in the Entity Inspector. You can revert the override to remove the override in order to resume inheriting from the source slice.
To revert property overrides

1. In the Entity Inspector, right-click the property, component, or entity that you modified.
2. Choose Revert overrides and choose one of the following:
   - **Property** – Revert changes to the property that you selected.
   - **Component** – Revert all changes to the component that you selected.
   - **Entity** – Revert all changes to the entity that you selected, such as added or removed components or changed properties.

Note

- In the Entity Inspector, reverting overrides to a parent entity doesn’t revert overrides to its children. This only affects changes that you made directly to the parent entity.
- If you revert overrides to the parent entity in Entity Outliner, this change also reverts overrides to its children.

Forcing a Property Override

Unmodified properties inherit changes made to the source slice. If you don’t want a property to inherit changes from the source slice, you can create an override to the property.

To force a property override

1. In the Entity Inspector, right-click an unchanged component property.
2. Choose Force property override.
The property override appears orange to indicate that it no longer inherits modifications from the slice.

### Slice Reloading

Slices support runtime reloading. If a slice has changed for any reason—such as a data push operation, retrieval from source control, or manual edit—Lumberyard Editor reloads the slice asset. Lumberyard Editor also recalculates any slice instances that were affected by the change.

### Working with Dynamic Slices

Slices are a powerful tool for organizing entity data in your worlds. In the editor, you can choose to cascade slices and organize entity data in any desired granularity and still receive the benefits of data sharing and inheritance throughout the hierarchy. A level-based game, for example, implements each level as its own slice asset that contains instances of many other slices. These slices can potentially cascade many levels deep. You can choose to create slices from other slices and inherit only the elements that you want.

Standard slice assets (.slice files) rely on the editor and cannot be instantiated at run time. However, Lumberyard provides a mechanism for designating any .slice asset that you've built as a dynamic slice. When you designate a slice as a dynamic slice, the Asset Processor processes and optimizes the slice for you, producing a .dynamicslice file asset. A dynamic slice is simply the runtime version of its source slice, containing only runtime components. The editor-dependent components have been converted to their runtime counterparts. Furthermore, dynamic slices no longer maintain a data hierarchy because doing so would increase memory footprint and reduce instantiation performance.

In the level-based game example previously mentioned, you could designate your giant level slice as a dynamic slice. When your game loads the level, it does so by instantiating the resulting .dynamicslice file.

You can choose to generate dynamic slices at whatever granularity is appropriate for your game. Because slices are loaded entirely asynchronously, they are a good choice for streaming strategies. For example, a
driving game might represent each city block as a separate slice and choose to load the slices predictively based on player driving behavior.

To create a dynamic slice

1. In the Asset Browser, right-click a .slice asset and choose Set Dynamic Slice.

2. Asset Processor processes the source .slice file and generates a .dynamicslice file. In the Asset Browser, you can see the .dynamicslice file appears as its own asset.

To remove a dynamic slice

- In the Asset Browser, right-click the dynamic file and choose Unset Dynamic Slice.

Asset Processor deletes the .dynamicslice file from the asset cache.

Instantiating Dynamic Slices

You can use the Spawner (p. 859) component to instantiate dynamic slices.

To instantiate dynamic slices

1. In the Asset Browser, select and then drag the .dynamicslice asset to the Slice property for the Spawner component.
2. To instantiate the dynamic slice when the level starts, select Spawn On Activate.
3. To instantiate the dynamic slice at a different time, you can use scripting or C++. The following Lua snippet uses an EBus call to tell the Spawner component to instantiate its dynamic slice:
SpawnerComponentRequestBus.Event.Spawn(self.entityId)

For more information about working with slices, see Working with Slices (p. 510).

Additional Links

- Amazon Lumberyard Getting Started series - Spawning dynamic slices (video)
- Spawning and Shooting a Projectile with Legacy Physics and Dynamic Slices with Lumberyard Script Canvas (video)

Slice Favorites

You can flag a slice as a favorite to make it easier to access and instantiate into your scenes. To use the slice favorites feature, enable the Slice Favorites (p. 1203) gem.

Slice favorites are:

- Saved per user and per project.
- Instantiated from the viewport or Entity Outliner.
- Flagged as a favorite from the Asset Browser.
- Stored in a directory structure that you can organize and rename to suit your work flow.
- Shared by importing or exporting.

Opening the Slice Favorites Panel

Your slice favorites appear in the Slice Favorites panel, where you can reorganize and manage them.

To open the Slice Favorites panel, do one of the following

- In the Entity Outliner, right-click to open the context menu, and choose Slice favorites, Manage favorites.

- In Lumberyard Editor, choose Tools, Other, Slice Favorites.

The Slice Favorites panel appears in its default location, above the Entity Inspector.
Flagging a Slice as a Favorite

You can flag a slice as a favorite so that your slice appears in the **Slice Favorites** panel.

**To flag a slice as a favorite**

- In the **Asset Browser**, do one of the following:
  - Right-click a slice and choose **Add as slice favorite**.
  - Drag a slice to the **Slice Favorites** panel.

Removing a Slice Favorite

You can remove a slice favorite so that it no longer appears in the **Slice Favorites** panel.

**To remove a slice favorite, do one of the following**

- In the **Asset Browser**, right-click the slice and choose **Remove as slice favorite**.
- In the **Slice Favorites** panel, right-click the slice, choose **Remove selected**, and then confirm the removal in the dialog.

Managing Slice Favorites

You can perform a variety of actions to manage your slice favorites in the **Slice Favorites** panel.

**To manage slices in Slice Favorites, you can do the following**

**Drag and Drop**

- Select one or more slices or folders to drag them into other folders or to reorder them in the list.
Add a folder

Right-click the Slice Favorites panel and choose Add folder.

Rename a folder

Right-click a folder in the Slice Favorites panel and choose Rename.

Rename a slice

Right-click a slice in the Slice Favorites panel, and choose Rename.

Note
Renaming a slice changes its display name in the Slice Favorites panel, but doesn't change the slice file name.

Import favorites from an XML file

Right-click the Slice Favorites panel and choose Import slice favorites.
Export favorites to an XML file

Right-click the Slice Favorites panel and choose Export slice favorites, and then select or create an XML to export.

Instantiating a Slice Favorite

To instantiate a slice favorite, do one of the following:

- Right-click the viewport or Entity Outliner, choose Slice favorites, and then navigate to the slice you want to instantiate.

- Drag a slice from the Favorite Slices panel into the viewport or the Entity Outliner.

Converting Slices with the Slice Upgrade Pipeline

In v1.23, Amazon Lumberyard introduced a new file format for component slices. Specifically, the override data found in data patches is now stored in a readable XML format instead of a hexadecimal byte stream. This is useful when performing diff and merge operations on your slice files, and enables a more robust versioning system for your serialized components. Use the Slice Upgrade Pipeline to automatically convert slice files to the latest slice file format.
Note
For more information about this new data patch format, and an example, see Anatomy of a Slice Data Patch (p. 515).

When do I use the Slice Upgrade Pipeline?

Use the Slice Upgrade Pipeline in Lumberyard v1.23 or later when:

• Adding a slice that was saved using an older version of the slice file format.
• Installing a new version, such as v1.23, or a build of Lumberyard that makes changes to the slice file format.
• Asset Processor logs indicate that your project has slices that require an upgrade. If so, the log displays a warning similar to the following:

This slice file is out of date: C:\lumberyard_version\dev\MyGame\slices\myslice.slice
To enable automatic upgrades:
In the settings file SliceBuilderSettings.json, set EnableSliceConversion to true and restart Asset Processor

Note
Currently, this pipeline does not update your slices when changing your component serialization. If you want to do that, use the Resave All Slices command on the File menu in Lumberyard Editor.

It is important to upgrade your slice assets as soon as possible, because:

• You can read, edit, and merge data patches more efficiently in upgraded slice assets.
• Upgrading your slices lets you use the new TypeChange and NameChange class builders when versioning your components. For more information about the component versioning system, see Versioning your Component Serialization (p. 1043).
• Delaying the upgrade puts you at a heightened risk of losing data in the future when changing serialization of your components. For more information about this risk, see Avoiding Data Loss when Serializing Component Data (p. 1049).

Note
In Lumberyard v1.23 and later, any slices with data patches still using the old format are updated automatically when changes are pushed to them. This is regardless of the slice conversion setting.

How do I convert my slices using the Slice Upgrade Pipeline?

Enable the Slice Upgrade Pipeline and restart Asset Processor to convert your slice files to the new format.

Perform the following steps:

1. Ensure that all source slice files are writable.
   • If you're using Perforce, recursively check out all the slice files that you want to upgrade by running the following Perforce command: p4 edit ...\*.slice
     You must change any read-only slices not tracked by Perforce to read-write manually.
2. Enable deep slice resaving in the file `SliceBuilderSettings.json`. This file is in your root dev directory. Open the file and change the setting `EnableSliceConversion` to true. By default, this option is disabled.

3. Restart Asset Processor.
   a. Close Lumberyard Editor.
   b. In the Windows taskbar, right-click Asset Processor, and choose Quit or press Ctrl+Q.
   c. Restart the project or branch that you are working on. Asset Processor automatically starts.

4. On the Tools tab in Asset Processor, trigger a Full Scan of assets.

![Asset Processor](image)

**Note**
For more information about using Asset Processor, see Using Asset Processor (p. 248).

5. Open Asset Processor logs. Confirm that there are no warning messages about slice files that are out of date or failed to convert.

After you perform these steps, while the Slice Upgrade Pipeline remains enabled, and Asset Processor is running, any slices that you copy or pull into your project are automatically converted by Asset Processor to the new format. A restart and full scan is not needed in this case.

**Troubleshooting the Slice Upgrade Pipeline**

Look at Asset Processor logs for slice files that failed conversion or display warnings. The following sections describe problems you might encounter.

**Warnings in the Slice Builder Settings File**

The Slice Upgrade Pipeline requires the settings file `SliceBuilderSettings.json` to be present in the `lumberyard_version\dev` folder. The default content of this file is:

```json
{
   "EnableSliceConversion": false
}
```

If there are problems with this file, one of the following warnings may appear in the Asset Processor log:

- Failed to Load Slice Builder Settings File. Using Default Slice Builder Settings.
• Slice Builder Settings File Missing. Using Default Slice Builder Settings.
• Loading Slice Builder Settings Failed. Using Default Slice Builder Settings.

Make sure the settings file exists in the correct location, is not open in another application, and contains the json content shown above.

Slice Dependency
If a nested slice is missing dependencies, the upgrade process fails because the slice instantiation failed. Correct the dependency problem to continue the upgrade.

Failed to Upgrade Slice: SlicePath - Slice Instantiation Failed.

Read-Only Files
Slices marked as read-only fail to process with the following log output:

This slice file is out of date: SlicePath
To Enabled Automatic Upgrades:
Make sure the slice isn't marked Read-Only. If using Perforce, check out the slice file.

Renaming the source slice file
During the upgrade process, if the source slice file cannot be renamed for any reason, one of the following errors may appear in the Asset Processor log:

• Failed to Upgrade Slice: SlicePath - Could not rename existing file
• Failed to Upgrade Slice: SlicePath - Could not rename new slice temp file
• Failed to Upgrade Slice: SlicePath - Could not open replacement slice file for writing
• Failed to Upgrade Slice: SlicePath - Could not write replacement slice file

If any of these errors appear, make sure the slice file is not open in another application, and that the file can be written to the directory where the source slice is located.

Component Reference

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Entity Inspector to add the following components to your entities.

Note
Some components are only available through gems. For example, you must enable the Rain (p. 1194) gem for your project, so that the Rain (p. 768) component is available. For more information, see Add modular features and assets with Gems (p. 1064).

AI

• Behavior Tree (p. 562)
Animation

- Actor (p. 536)
- AnimGraph (p. 545)
- Anim Graph Net Sync (p. 545)
- Attachment (p. 550)
- Simple Motion (p. 826)

Audio

- Audio Area Environment (p. 551)
- Audio Environment (p. 552)
- Audio Listener (p. 553)
- Audio Preload (p. 555)
- Audio Proxy (p. 557)
- Audio Rtpc (p. 557)
- Audio Switch (p. 558)
- Audio Trigger (p. 559)

Camera

- Camera (p. 566)
- Camera Rig (p. 570)

Editor

- Comment (p. 584)

Environment

- Fog Volume (p. 593)
- Infinite Ocean (p. 615)
- Lightning (p. 657)
- Lighting Arc (p. 667)
- Rain (p. 768)
- Sky Cloud (p. 830)
- Sky Highlight (p. 847)
- Snow (p. 853)
- Water Volume (p. 925)
Gameplay

- Input (p. 642)
- Random Timed Spawner (p. 776)
- Simple State (p. 828)
- Spawner (p. 859)
- Tag (p. 875)

Network

- Network Binding (p. 700)

PhysX

The following components support the PhysX system, which is not compatible with the legacy physics system. You can't use components interchangeably between each system.

For more information, see Simulating physics behavior with the PhysX system (p. 2772).

- PhysX Ball Joint component (p. 726)
- Cloth component (p. 575)
- Blast Family component (p. 564)
- Blast Family Mesh Data component (p. 566)
- PhysX Character Controller (p. 729)
- PhysX Collider (p. 735)
- PhysX Fixed Joint component (p. 745)
- PhysX Force Region (p. 746)
- PhysX Hinge Joint component (p. 755)
- PhysX Ragdoll (p. 757)
- PhysX Rigid Body (p. 759)
- PhysX Shape Collider (p. 763)
- PhysX Terrain (p. 767)

Physics (Legacy)

The components for the legacy physics system are not compatible with the PhysX system. For more information, see Physics in the Amazon Lumberyard Legacy Reference.

Rendering

- Area Light (p. 546)
- Decal (p. 585)
- Environment Probe (p. 587)
- Geom Cache (p. 600)
- High Quality Shadow (p. 614)
- Lens Flare (p. 654)
• Mesh (p. 684)
• OccluderArea (p. 701)
• Particle (p. 702)
• Point Light (p. 706)
• Portal (p. 717)
• Projector Light (p. 723)
• Render to Texture (p. 781)
• VisArea (p. 918)

Scripting
• Lua Script (p. 683)
• Script Canvas (p. 818)
• Trigger Area (p. 890)

Shape
• Box Shape (p. 818)
• Capsule Shape (p. 818)
• Compound Shape (p. 818)
• Cylinder Shape (p. 818)
• Polygon Prism Shape (p. 708)
• Sphere Shape (p. 818)
• Spline (p. 866)
• Tube Shape (p. 894)
• White Box component (p. 946)
• White Box Collider component (p. 960)

Terrain
• Legacy Terrain level component (p. 651)
• River (p. 796)
• Road (p. 808)

UI
• UI Canvas Asset Ref (p. 897)
• UI Canvas Proxy Ref (p. 898)
• UI Canvas on Mesh (p. 897)

VR
• VR Preview (p. 924)
Miscellaneous

- Transform (p. 878)

Actor

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Actor component to create characters for your game. After you import your character files from your DCC tool into Lumberyard, you can create an entity and add the Actor component to it. For example, you must use an Actor component to create a controllable character for your game.

For the Actor component to work properly, you must also add one of the following:

- Simple Motion (p. 826) component – Uses a single motion for your actor.
- AnimGraph (p. 545) component – Uses an animation graph to control your actor's behavior.

Topics

- Actor Component Properties (p. 536)
- Using Multiple Skin Attachments for an Actor (p. 537)
- Setting Up Actor Entities (p. 541)

Actor Component Properties

The Actor component has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor asset</td>
<td>Specifies the actor file that you want to add to your entity.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LOD Materials</td>
<td>Specifies the material that is linked to your actor asset.</td>
</tr>
<tr>
<td>Attachment type</td>
<td>The Actor component has the following attachment types:</td>
</tr>
<tr>
<td></td>
<td>• Actor Attachment – Attaches the current actor to the target entity.</td>
</tr>
<tr>
<td></td>
<td>• Skin Attachment – Attaches a skin attachment to a target entity.</td>
</tr>
<tr>
<td>Draw skeleton</td>
<td>Determines whether character joints are visible.</td>
</tr>
<tr>
<td>Draw character</td>
<td>Determines whether character mesh is visible.</td>
</tr>
<tr>
<td>Skinning method</td>
<td>Specifies the skinning method to use for the actor. You can choose the following options:</td>
</tr>
<tr>
<td></td>
<td>• Dual quat skinning – Specify dual quaternion skinning to help smooth deformation effects and preserve the volume of the mesh. Dual quaternion skinning is a technique that uses dual quaternions to provide a more visually pleasing appearance for blended vertex positions in a skinned mesh. This technique effectively substitutes an interpolation of rotation for the linear interpolation. Dual quaternion skinning preserves volumes and resists twists better than linear skinning.</td>
</tr>
<tr>
<td></td>
<td>• Linear skinning – Specify linear skinning to make a mesh follow a character’s joint. When a character flexes a joint, the mesh vertices move with the bones of the character’s animation skeleton. For example, a vertex in a flexible area like an elbow or knee calculates where it will be if the vertex moves with each bone that affects it and then linearly interpolates between them.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Linear skinning tends to collapse around joints with twists or large blend angles.</td>
</tr>
<tr>
<td></td>
<td>You can also enable linear skinning in Geppetto. For more information, see Enabling Linear Skinning with Geppetto in the Amazon Lumberyard Legacy Reference.</td>
</tr>
</tbody>
</table>

Using Multiple Skin Attachments for an Actor

You can use multiple skin attachments on an actor or character to add interchangeable parts such as a hat or belt. The meshes for these additional attachments can be a part of a single FBX file or separated into multiple FBX files. When deciding whether to use a single FBX file or multiple FBX files, consider the following:
Topics

• Using a Single FBX File (p. 538)
• Using Multiple FBX Files (p. 541)

Single FBX file with all skinned attachments (meshes) embedded

• Works well for an actor with a limited set of added or active attachments.
• Causes initial performance slowdown with Asset Processor, especially with high fidelity meshes. The end result performance, however, is the same as with multiple FBX files.
• Reduces number of FBX files to manage, but limits flexibility in modifying assets.

Multiple FBX files, each embedded with a skin attachment (mesh) that can be applied to an actor.

• Multiple source files to manage within a project.
• Provides flexibility for multiple content creators to manipulate meshes as needed.
• Asset Processor generates an .mtl (material) file for each FBX regardless of whether you intend to share materials.

Whether you use a single FBX file or multiple FBX files, you create an actor for each skinned mesh attachment. The single FBX file method requires some additional configuration in the FBX Settings tool.

For more information, see Customize FBX asset export with FBX Settings (p. 409).

To download the assets used in the following sections, see Amazon Lumberyard Downloads and choose the Lumberyard characters file.

Using a Single FBX File

When you use a single FBX file embedded with multiple meshes, you must create an actor for each attachment. For example, a cowboy character would be an actor, while his interchangeable hat and belt must each be added as additional actors.

To add multiple actors from a single FBX file in the FBX Settings tool

1. In Lumberyard Editor's Asset Browser, right-click the .fbx file that contains your meshes and choose Edit Settings.
2. On the Actors tab, click Add another actor.
   You should already have one actor for your character. The additional actors that you add are the attachments, such as articles of clothing.
3. For the Actor group, enter the actor name such as cowboy_hat and then click the mesh icon to select the mesh that you want as an attachment.
4. In the Select nodes dialog box, select the mesh, and then click Select.
Your **Actor** groups should look like the following.
Example
5. Click **Update** to save your settings and close the **FBX Settings** tool.

6. In the **Asset Browser**, navigate to the directory that contains your FBX file to view your actor files. You can see the primary actor and each attachment that you added.

**Example**

In the **Asset Browser**, a single FBX file contains the primary actor and the attachments.

![Asset Browser example](image)

**Using Multiple FBX Files**

When you use an FBX file for each skin attachment, Asset Processor automatically generates an **Actor** file for each FBX file. Modify the **FBX Settings** tool only if you want to change the default Asset Processor behavior.

**Example**

In the **Asset Browser**, multiple FBX files each contain a separate actor file.

![Additional Asset Browser example](image)

**Setting Up Actor Entities**

After you set up your skinned meshes as actors, you can create your actor entities. You then parent the attachments to the main entity and line the attachments up to the primary actor.
To create actor entities

1. From the **Asset Browser**, select and drag the main actor file and the attachment actor files into the viewport.

   Lumberyard automatically adds each file as its own actor component entity.

2. In the **Entity Outliner**, click and drag the attachment entities to the main entity; this parents the attachment actor entities to the main actor entity.

   **Note**
   The attachments entities may not line up with the main entity. You'll fix this in the next step.
Example

3. To align the children with its parent, select a child (attachment) entity and then in the Entity Inspector, specify the Translate values to 0. Repeat for any other child entities.
The attachments now line up with the main actor entities.

4. For each child entity, do the following:
   a. In the **Actor** component, for **Attachment type**, choose **Skin attachment**.
   b. For **Target entity**, click the picker icon and then select the primary actor to attach the skinned mesh (for example, the **cowboyactor**).
Your component entity setup is complete. When your primary actor animates, the additional skinned mesh attachments animate with the primary actor skeleton.

**AnimGraph**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the AnimGraph component to add an animation graph and motion set to your character. Add this component to the Actor (p. 536) component to control character behavior from an animation graph. For single motions, see the Simple Motion (p. 826) component.

**AnimGraph Component Properties**

The AnimGraph component has the following properties:

- **Anim graph**
  - Lets you select the animation graph that was created in the Animation Editor.

- **Motion set asset**
  - Lets you select the motion set that was created in the Animation Editor.

- **Active motion set**
  - Lets you select the motion set that was created in the Animation Editor.

- **Parameters**
  - Displays the parameters that were created with the Animation Graph.

**Anim Graph Net Sync**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The AnimGraph (p. 545) component, which adds an animation graph and motion set to a character, does not automatically synchronize its parameters across the network. Lumberyard's GridMate networking system (p. 1990) provides a server an authoritative way of replicating these parameters. This replication enables the movements of a character on a server to be mirrored on all of the clients that are connected to the server.

To implement the replication, use Lumberyard's Anim Graph Net Sync component, which is included with the EMotion FX Animation (p. 1124) gem. The Anim Graph Net Sync component requires the Network Binding (p. 700) component and can be added to any entity that has the AnimGraph component.

For more information, see Synchronizing Animations Across a Network (p. 2051).
Area Light

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Area Light component on an entity to light an area.

The Area Light component has the following settings:

Visible
- Shows the light.

On initially
- When created, the light is on by default.

General Settings

See the following general settings:

Color
- The color of the light.

Diffuse multiplier
- Sets the strength of the diffuse color.

Specular multiplier
- Sets the strength of the specular brightness.

Ambient
- Light acts as a multiplier for cubemap values.

Area Light Settings

See the following area light settings:

Area width
- Width of the area light in meters.

Area Height
- Height of the area light in meters.

Max Distance
- Maximum distance in meters that the area light extends.

FOV
- Field of View (FOV) in degrees.

Options

See the following options:
View distance multiplier

Adjusts the maximum view distance. For example, 1.0 uses the default and 1.1 is 10% farther than the default.

Minimum spec

Minimum specification value at which the light is enabled.

Cast shadow spec

The minimum specification at which shadows are cast.

Voxel GI mode

Mode for light interaction with voxel global illumination (GI). Choose None, Static, or Dynamic.

Use VisAreas

Light uses visible areas. If unselected, light ignores visible areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.

Volumetric fog

Light affects volumetric fog and surrounding area.

Shadow Settings

See the following shadow settings:

Terrain Shadows

Includes the terrain in the shadow casters for this light.

Animation

See the following animation settings:

Style

Enter a number to specify a preset light animation curve to play as defined in the Light.cfx file. Valid values are 0 to 48. You can also use values 40 to 48 for testing and debugging.

Speed

Multiple of the base animation rate. For example, a value of 2.0 makes an animation play twice as fast.

Phase

Animation start offset from 0 to 1. A value of 0.1 is 10% into the animation. For example, you can use this setting, for to prevent lights in the same scene, with the same animation, from being animated in unison.
EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

SetLightState

Turns the light on or off.

Parameters

On or Off

Return

None

Scriptable

Yes

TurnOnLight

Turns the light on.

Parameters

None

Return

None

Scriptable

Yes

TurnOffLight

Turns off the light.

Parameters

None

Return

None

Scriptable

Yes

ToggleLight

Toggles the light state from on to off, or off to on.
Area Light

Parameters

None

Return

None

Scriptable

Yes

EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information, see Working with the Event Bus (EBus) system (p. 1851).

LightTurnedOn

Sends a signal when the light is turned on.

Parameters

None

Return

None

Scriptable

Yes

LightTurnedOff

Sends a signal when the light is turned off.

Parameters

None

Return

None

Scriptable

Yes

The following is an example of script using the Request Bus Interface.

```lua
function example:OnActivate()
    LightComponentRequestBus.Event.TurnOnLight(self.entityId)
    LightComponentRequestBus.Event.TurnOffLight(self.entityId)
    LightComponentRequestBus.Event.ToggleLight(self.entityId)
end
```
Attachment

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/game/) or visit the [AWS Game Tech blog](https://aws.amazon.com/training/learning-journey/video-game-engineer/) to learn more.

The **Attachment** component lets an entity attach to a bone on the skeleton of another entity.

### Attachment Component Properties

![Attachment Component](image)

The **Attachment** component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target entity</td>
<td>Specifies the character entity that you want to attach.</td>
</tr>
<tr>
<td>Joint name</td>
<td>Specifies the joint that you want to attach to the entity.</td>
</tr>
<tr>
<td>Position offset</td>
<td>Specifies the x, y, and z local position offset from the target bone.</td>
</tr>
<tr>
<td>Rotation offset</td>
<td>Specifies the x, y, and z local rotation offset from the target bone.</td>
</tr>
<tr>
<td>Scale offset</td>
<td>Specifies the x, y, and z local scale offset from the target bone.</td>
</tr>
<tr>
<td>Attached initially</td>
<td>Specifies whether to attach to the target entity automatically.</td>
</tr>
<tr>
<td>Scaling</td>
<td>Specifies how the object scaling is determined. You can specify the following values:</td>
</tr>
<tr>
<td></td>
<td>• Use world scale – Attached object is scaled in world space.</td>
</tr>
<tr>
<td></td>
<td>• Use target entity scale – Attached object adopts scale of target entity.</td>
</tr>
<tr>
<td></td>
<td>• Use target bone scale – Attached object adopts scale of target entity/joint.</td>
</tr>
</tbody>
</table>

### EBus Request Bus Interface

Use the following request functions with the event bus (EBus) interface to communicate with other components of your game.

For more information, see Working with the Event Bus (EBus) system (p. 1851).
### Lumberyard User Guide

#### Audio Area Environment

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach</td>
<td>Changes the attachment target for an entity. The entity will detach from its previous target.</td>
<td>entityIdId – ID of the entity in which to attach. targetBoneName – Name of bone in which to attach the entity. If a bone is not found, then attach to target entity's transform origin. offsetTransform – Attachment's offset from target.</td>
<td>Yes</td>
</tr>
<tr>
<td>Detach</td>
<td>Detaches the target from the entity.</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>SetAttachmentOffset</td>
<td>Update entity's offset from target.</td>
<td>offsetTransform – Attachment's offset from target.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information, see [Working with the Event Bus (EBus) system](#) (p. 1851).

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnAttached</td>
<td>Indicates that the entity has attached to the target.</td>
<td>entityIdId – ID of the target in which is being attached.</td>
<td>Yes</td>
</tr>
<tr>
<td>OnDetached</td>
<td>Indicates that the entity is detaching from its target.</td>
<td>entityIdId – ID of the target in which is being detached.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Audio Area Environment

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Using the **Audio Area Environment** component, you can apply environment effects to sounds that an entity triggers. You must also add a shape component to use the audio area environment component.

**Audio Area Environment Properties**

The **Audio Area Environment** component has the following properties:
**Broad-phase Trigger Area**

Link this property to an entity that includes a trigger area and shape. This trigger area is used for broad-phase checks. The **Audio Area Environment** component tracks any entity that moves inside the trigger area.

Default: None

**Environment name**

The name of the Audio Translation Layer (ATL) (p. 3147) environment to apply to entities in the area.

Default: None

**Environment fade distance**

The distance around a shape where the environment amounts will fade based on an entity's distance from the shape. Only positive, non-zero values are valid.

Default: 1.0

---

**Using the Audio Area Environment Component**

Setting up the **Audio Area Environment** component requires two entities. The second entity is linked to the **Audio Area Environment** component in the first entity, acting as a broad-phase trigger area. When these entities are configured properly, any entity that passes near or through the inner shape of the first entity will have an environment amount applied to any triggered sounds.

**To set up the Audio Area Environment component**

1. In Lumberyard Editor, right-click the viewport in your level, and click **Create new component entity**.
2. Click **Tools**, **Entity Inspector**. Be sure that your new component entity is selected in the viewport.
3. In the **Entity Inspector**, click **Add Component**, **Shape**, and then select one of the shape options.
4. Click **Add Component**, **Audio**, **Audio Area Environment**.
5. Repeat steps 1 – 4 to add another entity with a **Shape** (any) component and a **Trigger Area** component (located under **Scripting**).
6. Place and size the two entities so that the following conditions are met:
   - The second entity's shape completely encompasses the first entity's shape.
   - The second entity's shape is larger than the first entity's shape by at least the value for the **Environment fade distance**. This allows the **Audio Area Environment** component to track an entity's distance from the inner shape when the entity enters the outer trigger area.
7. (Optional) In the **Entity Inspector**, for the **Trigger Area** component, use the **Tag Filters** to filter entities that you don't want the **Audio Area Environment** component to process.

---

**Audio Environment**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Audio Environment** component provides access to features of the Audio Translation Layer (ATL) (p. 3147) environments. Environments are used to apply environmental effects such as reverb or echo.
Audio Environment Properties

The Audio Environment component has the following property:

Default Environment

Enter the name of the audio environment to use by default when setting amounts.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

SetAmount

Sets the amount of environmental 'send' to apply to the default environment, if set.

Parameters

- amount – Float value of the amount to set

Return

- None

Scriptable

- Yes

SetEnvironmentAmount

Sets the amount of environmental 'send' to apply to the specified environment.

Parameters

- environmentName – Name of ATL Environment to set an amount on
- amount – Float value of the amount to set

Return

- None

Scriptable

- Yes

Audio Listener

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
With the **Audio Listener** component, you can place a virtual microphone in the environment. An audio listener acts as a sink for sound sources in the virtual world, and 3D audio rendering is processed with respect to the listener's world transform. You can specify the audio listener's position and rotation independently.

### Audio Listener Properties

The **Audio Listener** component has the following properties:

**Rotation Entity**

Link this property to an entity where the audio listener adopts the rotational part of the transform. The current entity is used if a value is not specified.

**Position Entity**

Link this property to an entity where the audio listener adopts the positional part of the transform. The current entity is used if a value is not specified.

**Fixed Offset**

Link this property to an entity where the audio listener adopts the offset part of the transform. The current entity is used if a value is not specified.

**Listener Enabled**

Controls the initial state of the listener.

### Using the Audio Listener Component

Only one audio listener is supported in a game. You can add the **Audio Listener** component to an entity that contains the game camera.

**To set up the Audio Listener component**

1. In Lumberyard Editor, right-click the viewport in your level, and click **Create new component entity**.
2. Click **Tools, Entity Inspector**. Be sure that your new component entity is selected in the viewport.
3. In the **Entity Inspector**, click **Add Component, Audio, Audio Listener**.

### EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**SetRotationEntity**

Specify the entity with the rotational part of the transform that the audio listener will adopt.

**Parameters**

- **entityId** – Entity to use for the rotational part of the transform

**Return**

None
Scriptable

Yes

SetPositionEntity

Specify the entity with the positional part of the transform that the audio listener will adopt.

Parameters

entityId – Entity to use for the positional part of the transform

Return

None

Scriptable

Yes

SetFullTransformEntity

Specify the entity with the full transform that the audio listener will adopt.

Parameters

entityId – Entity to use for the transform

Return

None

Scriptable

No

Audio Preload

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Using the Audio Preload component, you can load and unload ATL preloads, which contain references to soundbanks. You can specify the loading type as automatic or manual. The automatic loading type means that preloads load when the component activates and unload when it deactivates. The manual loading type means the component does not take any action until the user makes a request.

Audio Preload Properties

The Audio Preload component has the following properties:

Preload Name

Name of the default ATL Preload that this component loads or unloads. Modify this property to define a custom default ATL preload.

Default: Blank

Load Type

Set to Auto or Manual.
If set to **Auto**, preloads load when component activates and unload when component deactivates.
If set to **Manual**, preloads load and unload only when the user makes the request to the interface.
Default: Auto

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see *Working with the Event Bus (EBus) system (p. 1851).*

<table>
<thead>
<tr>
<th>Load</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Load</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Loads the default ATL preload (if it is set)</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Scriptable</strong></td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Unload</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Unloads the default ATL preload (if it is set)</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Scriptable</strong></td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>LoadPreload</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Loads an ATL preload by name</td>
</tr>
</tbody>
</table>
| **Parameters** | *preloadName*  
Name of an ATL Preload to load |
| **Return** | None |
| **Scriptable** | Yes |

<table>
<thead>
<tr>
<th>Load</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>UnloadPreload</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Unloads an ATL preload by name</td>
</tr>
</tbody>
</table>
Audio Proxy

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Audio Proxy component is a required dependency if you add multiple audio components to an entity. It acts as a proxy audio object wrapped in a component. For example, if you have an audio trigger component and an audio rtpc component on the same entity, they communicate to the same audio object using this audio proxy component.

Audio Rtpc

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Audio RTPC component provides basic Real-Time Parameter Control (RTPC) (p. 3147) functionality. An RTPC is a named variable that the audio system can interpret in many different ways. It allows game developers to set the value from the game at run time to produce real-time tweaking of sounds.

Audio RTPC Component Properties

The Audio RTPC component has the following property:

Default Rtpc

Enter the name of the audio RTPC to use by default. You can associate any RTPC name with the entity, typically one that is meant to affect a particular trigger.
EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

SetValue

Sets the value of the default RTPC.

Parameters

value – Float value of the RTPC

Return

None

Scriptable

Yes

SetRtpcValue

Sets the value of the specified RTPC.

Parameters

rtpcName – Name of the RTPC to set
value – Float value to set

Return

None

Scriptable

Yes

Audio Switch

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Audio Switch component provides basic Audio Translation Layer (ATL) (p. 3147) switch functionality. With switches (and switch states), you can specify the state of an entity. The audio middleware interprets states, modifies the behavior of sounds, and plays the appropriate sounds.

Audio Switch Properties

The Audio Switch component has the following properties:

Default Switch

Enter the name of the audio switch to use by default. You can associate any audio switch with the entity.
Default State

Enter the name of the audio switch state to use by default. Use the Audio Controls Editor (p. 3142) to assign the state to the switch. When this component is activated, the default switch is set to the default state.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

SetState

Sets the specified state of the default switch.

Parameters

- stateName – Name of the state to set

Return

- None

Scriptable

- Yes

SetSwitchState

Sets a specified switch to a specified state.

Parameters

- switchName – Name of the switch to set

- stateName – Name of the state to set

Return

- None

Scriptable

- Yes

Audio Trigger

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Audio Trigger component provides basic play and stop features so that you can set up Audio Translation Layer (ATL) (p. 3147) play and stop triggers that can be executed on demand. With an audio trigger, you can also enable the player to run or stop audio triggers by name on entities.
Audio Trigger Properties

The Audio Trigger component has the following properties.

Default 'play' Trigger

Enter the name of the audio trigger that this component runs when 'play' is called. You can change this property to specify a different default audio trigger.

Default 'stop' Trigger

Enter the name of the audio trigger that this component runs when 'stop' is called. You can specify any trigger here; you do not need to specify a 'stop' trigger in order to stop audio, but it is a best practice to pair the two triggers. If you leave this setting blank, the 'stop' trigger simply stops the audio trigger specified for 'play'.

Obstruction Type

Select an option for the raycasts used in calculation of obstruction and occlusion.

• Ignore
• SingleRay
• MultiRay

Play immediately

Select this option to run upon component activation the audio 'play' trigger.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

Play

Runs the default 'play' trigger, if set.

Parameters

None

Return

None

Scriptable

Yes

Stop

Runs the default 'stop' trigger, if set. If no 'stop' trigger is set, ends the default 'play' trigger.

Parameters

None
Audio Trigger

ExecuteTrigger
Runs the specified audio trigger.

Parameters
triggerName – Name of the audio trigger to run.

Return
None

Scriptable
Yes

KillTrigger
Cancels the specified audio trigger.

Parameters
triggerName – Name of the audio trigger to cancel.

Return
None

Scriptable
Yes

KillTrigger
Cancels all audio triggers that are active on an entity.

Parameters
None

Return
None

Scriptable
Yes

SetMovesWithEntity
Specifies whether triggers should update position as the entity moves.

Parameters
shouldTrackEntity – Boolean indicating whether triggers should track the entity's position.
EBus Response Bus Interface

Use the following response functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

OnTriggerFinished

Informs all listeners about an audio trigger that has finished playing (the sound has ended).

Parameters

- triggerId – ID of trigger that was successfully executed.

Return

- None

Scriptable

- Yes

Behavior Tree

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Behavior Tree component to load and run a behavior tree for the attached entity.

Behavior Tree Component Properties

The Behavior Tree component has the following properties:

Behavior tree asset

- Select an XML file that contains a behavior tree definition.

Enabled initially

- When selected, the behavior tree is loaded and activated with the entity.

EBus Request Bus Interface

Use the following request functions with the event bus (EBus) interface, BehaviorTreeComponentRequestBus, to communicate with other components of your game.
For more information about using the EBus interface, see Working with the Event Bus (EBus) system (p. 1851).

StartBehaviorTree

Starts an inactive behavior tree associated with this entity.

Parameters

None

Return

None

Scriptable

Yes

StopBehaviorTree

Stops an active behavior tree associated with this entity.

Parameters

None

Return

None

Scriptable

Yes

GetVariableNameCrcs

Gets a list of all crc32s of the variable names.

Parameters

None

Return

AZStd::vector<AZ::Crc32>

Scriptable

Yes

GetVariableValue

Gets the value associated with a variable.

Parameters

AZ::Crc32 variableNameCrc

Return

bool
Scriptable

Yes

SetVariableValue

Sets the value associated with a variable.

Parameters

AZ::Crc32 variableNameCrc

bool newValue

Return

None

Scriptable

Yes

The following is an example of script using the Request Bus Interface.

```lua
local behaviortreescript =
{
    Properties =
    {
        Target = EntityId(),
    },
}

function behaviortreescript:OnActivate()
    BehaviorTreeComponentRequestBus.Event.StartBehaviorTree(self.entityId)
    BehaviorTreeComponentRequestBus.Event.SetVariableValue(self.entityId,
        Crc32("HasTarget"), self.Properties.Target:IsValid())
end

return behaviortreescript
```

Blast Family component

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With the Blast Family component, you can enable destruction simulation using the NVIDIA Blast library, and set properties for the simulation. The Blast Family component is used with the Blast Family Mesh Data component. This topic describes the properties of the Blast Family component.

The Blast Family component is provided by the NVIDIA Blast gem (p. 1181).

For information on using the Blast Family component see Simulated destruction with NVIDIA Blast (p. 2833).
Blast Family component properties

Blast Asset

The blast asset that will be used for the destruction simulation.

Blast Material

A blast material from the blast material library. Blast materials define how much damage various forces cause to the bonds holding the fractured asset together, and how much damage is required to trigger destruction. For more information, see Specify destruction properties with Blast materials (p. 2851).

Physics Material

The physics material for the blast asset. Physics materials define physical properties such as friction.

Collision Layer

The collision layer for this Blast Family.

Collides With

The collision group containing the layers that this Blast Family collides with.

Simulated

When enabled, this Blast Family's collision will be part of the PhysX simulation.

In Scene Queries

When enabled, this Blast Family's colliders will be available for scene queries.

CCD Enabled

When enabled, this Blast Family will use continuous collision detection. CCD is useful for ensuring accurate collision detection for high-speed objects.

Tag

Set a tag for this Blast Family. Tags can be used to quickly identify components in script or code.
Blast Family Mesh Data component

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With the Blast Family Mesh Data component, you can set the mesh and material assets for NVIDIA Blast entities. The Blast Family Mesh Data component is used with the Blast Family component. This topic describes the properties of the Blast Family Mesh Data component.

The Blast Family Mesh Data component is provided by the NVIDIA Blast gem (p. 1181).

For information on using the Blast Family Mesh Data component see Simulated destruction with NVIDIA Blast (p. 2833).

Blast Family Mesh Data component properties

Show mesh assets

When enabled, you can manually add mesh assets to this Blast Family Mesh Data component.

The Show mesh assets property exposes the Mesh assets list. Choose the + button to the right of Mesh assets to add a list entry. Choose the Folder button to select a mesh for the list entry.

Material

The material that supplies the visual appearance for the meshes in this Blast Family Mesh Data component.

Blast Slice

A blast slice for this Blast Family Mesh Data component. Blast slices are generated when blast assets are processed by the Python asset builder for NVIDIA Blast. A blast slice will automatically add the mesh assets and material to the Blast Family Mesh Data component.

Camera

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The **Camera** component allows an entity to be used as a camera. To use the **Camera** component, you must first add a Camera Framework gem to your project. For information, see [Camera Framework Gem](p. 1121).

**Camera Component Properties**

![Camera component properties](image)

The **Camera** component has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field of view</td>
<td>Vertical field of view in degrees.</td>
<td>75.0 degrees</td>
</tr>
<tr>
<td>Near clip distance</td>
<td>Distance to the near clip plane of the view frustum in meters.</td>
<td>0.2 m</td>
</tr>
<tr>
<td>Far clip distance</td>
<td>None</td>
<td>1024.0 m</td>
</tr>
<tr>
<td>Frustum length</td>
<td>Frustum length percentage.</td>
<td>1.0 percent</td>
</tr>
<tr>
<td>Frustum color</td>
<td>Frustum color.</td>
<td>255,255,0</td>
</tr>
</tbody>
</table>
**Description**

**Distance clip distance**

near far plane of the view frustum in meters.

Default value: 1024

**Editor this camera**

selected camera as its view.

For more information, see *Changing the Camera View* (p. 209).

**Frustum length**

the frustum shape.

Default value: 1.0 percent
### Description

**Frustum color**

the frustum shape.

Default value: 255, 255, 0

### EBus Request Bus Interface

Use the following request functions with the event bus (EBus) interface, CameraRequestBus, to communicate with other components of your game.

For more information, see Working with the Event Bus (EBus) system (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFov</td>
<td>Gets the current field of view.</td>
</tr>
<tr>
<td>SetFov</td>
<td>Sets the current field of view.</td>
</tr>
<tr>
<td>GetNearClipDistance</td>
<td>Gets the current near clip distance.</td>
</tr>
<tr>
<td>SetNearClipDistance</td>
<td>Sets the current near clip distance.</td>
</tr>
<tr>
<td>GetFarClipDistance</td>
<td>Gets the current far clip distance.</td>
</tr>
<tr>
<td>SetFarClipDistance</td>
<td>Sets the current far clip distance.</td>
</tr>
</tbody>
</table>

The following is an example of script using the **Request Bus Interface**.

```lua
local camerasample = {
    Properties = {
    },
}

function camerasample:OnActivate()
    CameraRequestBus.Event.SetFov(self.entityId, 85)
    local nearClip = CameraRequestBus.Event.GetNearClipDistance(self.entityId)
    CameraRequestBus.Event.SetFarClipDistance(self.entityId, nearClip + 1024)
end

return camerasample
```

### Creating Camera Entity from View

You can create a static camera view from a specific entity by right-clicking an entity in the viewport and choosing **Create camera entity from view**. This places a new entity with a camera component at the same point. You can adjust the view of the camera by modifying its transform component.
Camera Rig

Use the **Camera Rig** component to add and remove behaviors to drive your camera entity. To use the camera component, you must first add the Camera Framework Gem (p. 1121) to your project.

**Camera Rig Component Properties**

The **Camera Rig** component has the following properties:

**Target acquirers** (p. 570)

Array of behaviors that define how a camera selects a target. The rig tries each acquirer in the order listed until one successfully finds a target.

**Look-at behaviors** (p. 571)

Array of behaviors that modify the look-at target transform. The rig runs each in order to generate a final target transform.

**Transform behaviors** (p. 573)

Array of behaviors that modify the camera transform based on the look-at target transform. The rig runs each in order before setting the camera component's transform.

**Target Acquirers**

**Target Acquirers** identify valid targets and acquire their transforms for use in other rig behaviors.

**Acquire By Tag**

**AcquireByTag** has the following properties:

**Target tag**

Find a target by tag. If multiple entities are found, it uses the first to respond.

**Use Target Rotation**

If selected, uses the target's rotation when determining camera behavior.

**Use Target Position**

If selected, uses the target's position when determining camera behavior.
Acquire By Entity Id

AcquireByEntityId has the following properties:

Entity Target

Select a specific entity to use as the camera target

Use Target Rotation

If selected, uses the target's rotation when determining camera behavior.

Use Target Position

If selected, uses the target's position when determining camera behavior.

Look-at Behaviors

Look-at Behaviors changes the target transform to modify camera behavior.

OffsetPosition

Use OffsetPosition to change the position of the target's transform. Positions are often determined from the base of a model. But suppose, for example, that you want to determine its position 1.8 meters up from its base. You can use this property to achieve that positional offset.

Look-at Behaviors has the following properties:

Positional Offset

Vector displacement of the target transform's position.

Offset Is Relative

If selected, uses local coordinates. If deselected, uses world-basis vectors for the offset.
**Rotate Camera Target**

Use **Rotate Camera Target** to rotate the target separately from its source target. For example, you may want your character to look up and down without pitching.

**Rotate Camera Target** has the following properties:

- **Axis of Rotation**
  
  The target cardinal's axis around which the camera rotates. Select the X, Y, or Z axis.

- **Event Name**
  
  Name of event that provides the values for the rotation.

- **Invert Axis**
  
  If selected, inverts the axis of rotation.

- **Rotation Speed Scale**
  
  Multiplier for new input values to scale the speed of rotation.

**SlideAlongAxisBasedOnAngle**

Use **SlideAlongAxisBasedOnAngle** to modify the position of the look-at target based on an angle. For example, say that you set the target to slide along the forward and backward axis based on pitch. As the target pitched down, then the position would move ahead of the target. If the target is attached to the character, then every time the target looked down, it would be ahead of the character. Every time it looked up, it would be behind the character.

**SlideAlongAxisBasedOnAngle** has the following properties:
Axis to slide along

Select an axis along which the target slides:
- Forwards and Backwards
- Right and Left
- Up and Down

Angle Type

Select an angle type on which to base the slide:
- Pitch
- Yaw
- Roll

Vector Component to Ignore

Select a vector component to ignore: None, X, Y, or Z.

Max Positive Slide Distance

The maximum slide along the axis when the angle reaches 90 degrees.

Max Negative Slide Distance

The maximum slide along the axis when the angle reaches -90 degrees.

Transform Behaviors

Transform Behaviors are a critical component of how the camera responds to the target. For example, you can set the camera to face the target, follow from a distance, or follow the target at a specific angle.

FaceTarget

FaceTarget causes the camera to change the rotation of its transform to look at the target. To use this feature, simply add it. There are no additional properties to configure.

FollowTargetFromAngle

FollowTargetFromAngle causes the camera to follow the target from a specified angle. This feature works well for top-down, isometric, and side scrolling cameras.

Follow Target from Angle has the following properties:

Angle

Angle at which to follow the target.
Rotation Type

Rotation type of the angle for following the target: yaw, pitch, or roll.

Distance from Target

The distance in meters from which the camera follows the target.

FollowTargetFromDistance

FollowTargetFromDistance causes the camera to follow the target from a specified distance. You can also set named events to trigger the camera to zoom in on or out from a target.

FollowTargetFromDistance has the following properties:

Follow Distance

The distance in meters from which the camera follows the target.

Minimum Follow Distance

Minimum distance from which the camera follows the target.

Maximum Follow Distance

Maximum distance from which the camera follows the target.

Zoom In Event Name

Event name that reduces the current follow distance, in effect zooming in.

Zoom Out Event Name

Event name that increases the current follow distance, in effect zooming out.

Zoom Speed Scale

Scale amount for the incoming zoom value.

Player Index

The index of the player (device index) that this feature supports.

Offset Position

Offset Position sets the camera's position to the target's position with an offset.
**Offset Position** has the following properties:

**Offset**

The vector offset in meters from the target.

**Is Offset Relative**

If selected, local basis vectors are used. If deselected, world basis vectors are used.

**Rotate**

Use **Rotate** to rotate a camera about one of its axes (X, Y, or Z).

**Rotate** has the following properties:

**Angle**

Angle in degrees to rotate the camera.

**Axis**

Axis about which to rotate the camera.

**Cloth component**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Cloth** component treats the vertices of any mesh that it references as cloth particles and applies physical properties, forces, and constraints to simulate the behavior of cloth. You can add this
component to any entity that has Mesh or Actor components. You can add multiple cloth components to an entity.

The Cloth component is provided by the NVIDIA Cloth gem (p. 1182).

For information on using the Cloth component see Simulate cloth with NVIDIA Cloth (p. 2857).

**Cloth component property groups**

- Base properties (p. 576)
- Motion constraints properties (p. 577)
- Backstop properties (p. 578)
- Damping properties (p. 578)
- Inertia properties (p. 579)
- Wind properties (p. 579)
- Collision properties (p. 580)
- Self Collision properties (p. 581)
- Fabric stiffness properties (p. 581)
- Fabric compression properties (p. 582)
- Fabric stretch properties (p. 583)
- Tether constraints properties (p. 583)
- Quality properties (p. 584)

**Base properties**
**Simulate in editor**

Enable to simulate cloth in editor.

**Mesh node**

The mesh node to simulate as cloth. The meshes available in the list have Cloth modifiers applied in FBX Settings.

**Mass**

Scale multiplier applied to the mass of all cloth particles. A value of 0.0 in this property makes all cloth particles static.

**Custom Gravity**

Enable to override global gravity and set a custom gravity value for this cloth.

**Gravity**

Set gravity for this cloth. The default, -9.81 on the Z axis, is standard gravity.

*Note*

The Gravity property is enabled by the Custom Gravity property.

**Gravity Scale**

Scale multiplier applied to the gravity of cloth particles.

**Stiffness frequency**

Stiffness exponent that adjusts the overall stiffness of the cloth simulation. This exponent is applied per second to these properties:

- Damping
- Damping - Linear drag
- Damping - Angular drag
- Wind - Air drag coefficient
- Wind - Air lift coefficient
- Self collision - Stiffness
- Fabric stiffness
- Fabric compression
- Fabric stretch
- Tether - Stretch

**Motion constraints properties**

<table>
<thead>
<tr>
<th>Motion constraints</th>
<th>Max Distance</th>
<th>Scale</th>
<th>Bias</th>
<th>Stiffness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0 m</td>
<td>1.0</td>
<td>0.0 m</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Max distance**

Maximum distance limit in meters for cloth particle movement.

**Scale**

Scale value applied to all motion constraints.
0.0: Motion constraints have no effect.
1.0: Motion constraints are fully applied.

Bias
Bias value in meters added to all motion constraints. Valid values range from -$\text{Infinity}$ to $\text{Infinity}$.

Stiffness
Stiffness for motion constraints.
0.0: Stiffness is not applied to motion constraints.
1.0: Stiffness is fully applied to motion constraints.

**Backstop properties**

<table>
<thead>
<tr>
<th>Backstop</th>
<th>0.1 m</th>
<th>0.0 m</th>
<th>0.0 m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Back offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Front offset</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note**
Backstop properties are only available when a Backstop vertex color stream is specified in the Cloth modifier for the selected Mesh node in FBX Settings.

Radius
The radius in meters of the backstop sphere.

Back offset
The offset in meters for backstop spheres behind the cloth.

Front offset
The offset in meters for backstop spheres in front of the cloth.

**Damping properties**

<table>
<thead>
<tr>
<th>Damping</th>
<th>X 0.2</th>
<th>Y 0.2</th>
<th>Z 0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear drag</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Angular drag</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Damping
Damping of cloth particle velocity.
0.0: Velocity is unaffected.
1.0: Velocity is zeroed.

Linear Drag
Portion of velocity applied to cloth particles.
0.0: Cloth particles are unaffected.
1.0: Damped global cloth particle velocity.

Angular Drag

Portion of angular velocity applied to turning cloth particles.

0.0: Cloth particles are unaffected.
1.0: Damped global cloth particle angular velocity.

Inertia properties

Linear

Portion of linear acceleration applied to cloth particles.

0.0: Cloth particles are unaffected.
1.0: Physically correct linear acceleration.

Angular

Portion of angular acceleration applied to turning cloth particles.

0.0: Cloth particles are unaffected.
1.0: Physically correct angular acceleration.

Centrifugal

Portion of angular velocity applied to turning cloth particles.

0.0: Cloth particles are unaffected.
1.0: Physically correct angular velocity.

Wind properties

Note

The component wind properties create wind that effects only the cloth referenced by the component. To create wind that can effect multiple components across multiple entities, see Create global or localized wind forces with PhysX (p. 2822).
Note
Wind is disabled when both the below Air drag and Air lift coefficients are 0.0.

Enable local wind velocity

Enable to set a wind Local velocity for this cloth. When disabled, the velocity from Physics::WindBus will be used.

Local velocity

Wind vector (direction and magnitude) in world coordinates. A greater magnitude applies a stronger wind force.

Note
The wind Local velocity property is enabled by the Enable local wind velocity property.

Air drag coefficient

Specifies how much drag air applies to the cloth particles.

Air lift coefficient

Specifies how much lift air applies to the cloth particles.

Air density

The density of air used for drag and lift calculations.

Collision properties

Friction

Controls the amount of friction between cloth particles and colliders.

0.0: Friction disabled.

Mass scale

Controls how quickly cloth particle mass is increased during collisions.

0.0: Mass scale disabled.

Continuous detection

Continuous collision detection improves collision by computing the time of impact between cloth particles and colliders.

Warning
The increase in quality can impact performance. We recommend that you use Continuous detection only when necessary.

Affects static cloth particles

Enable to allow colliders to move static cloth particles. Static cloth particles have a 0.0 inverse mass.
Self Collision properties

Distance

The minimum distance that the colliding cloth particles must maintain from each other in meters.

0.0: Self collision disabled.

Stiffness

Stiffness for the self collision constraints.

0.0: Self collision disabled.

Fabric stiffness properties

Horizontal

Stiffness value for horizontal stretch and compression constraints.

0.0: No horizontal stretch and compression constraints.

Horizontal multiplier

Scale value for horizontal stretch and compression constraints.

0.0: No horizontal stretch and compression constraints applied.

1.0: Fully apply horizontal stretch and compression constraints.

Vertical

Stiffness value for vertical stretch and compression constraints.

0.0: No vertical stretch and compression constraints.

Vertical multiplier

Scale value for vertical stretch and compression constraints.

0.0: No horizontal stretch and compression constraints applied.
1.0: Fully apply horizontal stretch and compression constraints.

**Bending**

Stiffness value for bending constraints. This value defines how easily a cloth folds on itself.

0.0: No bending constraints.

**Bending multiplier**

Scale value for bending constraints.

0.0: No bending constraints applied.

1.0: Fully apply bending constraints.

**Shearing**

Stiffness value for shearing constraints. This value defines how easily a cloth twists.

0.0: No shearing constraints.

**Shearing multiplier**

Scale value for shearing constraints.

0.0: No shearing constraints applied.

1.0: Fully apply shearing constraints.

---

**Fabric compression properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal limit</td>
<td>0.0</td>
</tr>
<tr>
<td>Vertical limit</td>
<td>0.0</td>
</tr>
<tr>
<td>Bending limit</td>
<td>0.0</td>
</tr>
<tr>
<td>Shearing limit</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Horizontal limit**

Compression limit for horizontal constraints. This property is affected by **Horizontal multiplier** in the **Fabric stiffness** property group.

0.0: Horizontal compression disabled.

**Vertical limit**

Compression limit for vertical constraints. This property is affected by **Vertical multiplier** in the **Fabric stiffness** property group.

0.0: Vertical compression disabled.

**Bending limit**

Compression limit for bending constraints. This property is affected by **Bending multiplier** in the **Fabric stiffness** property group.

0.0: Bending compression disabled.

**Shearing limit**

Compression limit for shearing constraints. This property is affected by **Shearing multiplier** in the **Fabric stiffness** property group.
0.0: Shearing compression disabled.

Fabric stretch properties

Note
For Fabric stretch properties, reduce Stiffness of Tether constraints or increase its Scale to allow cloth to stretch.

<table>
<thead>
<tr>
<th>Fabric stretch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal limit</td>
</tr>
<tr>
<td>Vertical limit</td>
</tr>
<tr>
<td>Bending limit</td>
</tr>
<tr>
<td>Shearing limit</td>
</tr>
</tbody>
</table>

Horizontal limit
Stretch limit for horizontal constraints. This property is affected by Horizontal multiplier in the Fabric stiffness property group.

0.0: Horizontal stretch disabled.

Vertical limit
Stretch limit for vertical constraints. This property is affected by Vertical multiplier in the Fabric stiffness property group.

0.0: Vertical stretch disabled.

Bending limit
Stretch limit for bending constraints. This property is affected by Bending multiplier in the Fabric stiffness property group.

0.0: Bending stretch disabled.

Shearing limit
Stretch limit for shearing constraints. This property is affected by Shearing multiplier in the Fabric stiffness property group.

0.0: Shearing stretch disabled.

Tether constraints properties

<table>
<thead>
<tr>
<th>Tether constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stiffness</td>
</tr>
<tr>
<td>Scale</td>
</tr>
</tbody>
</table>

Stiffness
Stiffness for tether constraints.

0.0: Tether constraints disabled.

1.0: Tether constraints behave like springs.
**Scale**

Scale factor for tether constraint **Stiffness**

**Quality properties**

- **Solver frequency**
  - Target solver iterations per second. The executed number of iterations per second may vary dependent on many performance factors. However, at least one iteration per frame is solved regardless of the value set.

- **Acceleration filter iterations**
  - Number of iterations to average the delta time factor used for gravity and external acceleration.

- **Remove static triangles**
  - Enable to remove triangles composed of static cloth particles. Enabling this property improves performance, however the removed static cloth particles will not be present for collision and self collision calculations.

**Comment**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Comment** component allows you to add long-form text comments for component entities. When enabled, the **Comment** component displays a dialog box that expands based on the size of the comment that you enter. The following examples demonstrate how you can use the comment text box:

- Explain how the scripts or components on an entity interact with other scripts or components
- Describe how everything in a level ties together
- Send descriptions, instructions, or notes to team members
Comment Properties

The Comment component has the following property:

Comment text box

Stores the user comment for the component entity.

Default: None

Using the Comment Component

You can use this feature by adding the component to an entity in your level.

To use the Comment component

1. In Lumberyard Editor, right-click the viewport in your level, and click Create entity.
2. In the Entity Inspector, click Add Component.
3. Under Editor, click Comment.
4. In the Entity Inspector, under Comment, add comments for the component entity in the text box.

Decal

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Decal component to place a component on an entity.

Decal Component Properties

The Decal component has the following properties:

Visible

If selected, shows the decal.

Projection Type

Specifies the type of decal projection: Planar, On Terrain, or On Terrain and Static Objects.

Default value: Planar

Material

The decal's material file.

Sort Priority

Sort priority relative to other decals in the system.

Valid values: 0 – 255

Default value: 16
Decal

**Depth**
Projection depth for deferred decals.
Valid values: 0 – 10
Default value: 1

**Offset**
Allows offsetting the decal relative to the entity's position.
Default value: 0,0,0

**Opacity**
Degree of transparency for the decal (only available for deferred decals).

**Deferred**
Shows No for Planar and On Terrain Projection Type. Shows Yes for On Terrain and Static Objects.

**Max view distance**
The furthest distance at which this decal can be viewed.

**View distance multiplier**
Multiplier to the automatically computed fade-out camera distance.

**Minimum spec**
Minimum spec for the decal to be active.

**EBus Request Bus Interface**
Use the following request function with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**SetVisibility**
Sets an explicit value (true/false) on the decal's visibility.

**Parameters**
- true or false

**Show**
Shows the decal.

**Parameters**
- None

**Hide**
Hides the decal.
Environment Probe

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Environment Probe component to achieve the right visual quality for a space. Environment probes help to determine proper reflections, ambient diffuse values, particle diffuse values, and shadow colors.

For information about how to blend environment probes to create a smooth transition, for example, from day to night, see Blending Environment Probes (p. 1252).

The Environment Probe component has the following settings:

Visible
  Shows the light.

On initially
  Specify if the light is on when created.

General Settings

See the following general settings:

Color
  The color of the light.

Diffuse multiplier
  Sets the strength of the diffuse color.

Specular multiplier
  Sets the strength of the specular brightness.

Environment Probe Settings

See the following environment probe settings:

Area dimensions
  The width, height, and depth in meters of the environment probe's effective area.
Sort priority

Priority number for probe rendering. Lower priority numbers (for example, 0 or 1) are rendered on top of the higher priority numbers (for example, 100).

Resolution

Cubemap resolution in pixels that is generated by this probe.

Box projected

Enables box projection. Does not require regeneration of the cubemap.

Box height

Height of the box projection.

Box length

Length of the box projection.

Box width

Width of the box projection.

Attenuation falloff

The value that the light begins to falloff. For example, a value or 1.0 means the light starts to fall off at 1.0 meter.

Options

See the following options:

View distance multiplier

Adjusts the maximum view distance. For example, 1.0 uses the default and 1.1 is 10% farther than the default.

Minimum spec

Minimum specification value at which the light is enabled.

Voxel GI mode

Mode for light interaction with voxel global illumination (GI).

Use VisArea

Light is affected by visible areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.

Volumetric fog

Light affects volumetric fog and surrounding area.
**Animation**

See the following animation settings:

**Style**

Enter a number to specify a preset light animation curve to play as defined in the `Light.cfx` file. Valid values are 0 to 48. You can also use values 40 to 48 for testing and debugging.

**Speed**

Multiple of the base animation rate. For example, a value of 2.0 makes an animation play twice as fast.

**Phase**

Animation start offset from 0 to 1. A value of 0.1 is 10% into the animation. For example, you can use this setting to prevent lights in the same scene, with the same animation, from being animated in unison.

**Cubemap Generation**

See the following cubemap generation settings:

**Cubemap**

Click **Generate** to generate or regenerate a cubemap for this probe. The button is enabled only when **Use customized cubemap** is deselected.

Click **Reset** to reset a cubemap for this probe.

**View cubemap**

Displays a preview of the cubemap in the viewport. You can only select this option when a cubemap exists.

**Use customized cubemap**

If selected, you can specify a custom cubemap for **Cubemap asset**. If deselected, a cubemap is generated.

**Cubemap asset**

File path for the cubemap asset.

**Additional Resources**

For more information about the **Environment Probe** component, see the following:

- Intro to Environment Lighting
- Getting Started: Using Lighting Tutorial (video)

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).
All light components share common EBus functions. For more information, see Light Components EBus Request Bus Interface (p. 548).

FastNoise Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **FastNoise Gradient** component generates gradient signals with pseudorandom values. The values are mapped to the world positions in the game project. It uses the third-party, open source FastNoise library, which provides a collection of noise algorithms to generate different noises. You can interpret the values to produce a variety of game effects throughout the world. For example, you can use this component with the Vegetation Gem to randomly scatter vegetation throughout your world.

The generated gradient signals can be used with other components provided by the **Gradient Signal** gem.

The **FastNoise Gradient** component is provided by the section called “Fast Noise Gem” (p. 1127).

Properties

The **FastNoise Gradient** component has the following properties. The properties listed varies depending on the **Noise Type** property.

**Preview and Preview Settings**

A preview image of the gradient signal.

**Noise Type**

The type of pseudorandom algorithm used to generate noise.

- **Value**: First, this algorithm generates noise in a similar way as the **White Noise** noise type. Then, each value is interpolated with other surrounding values. The result looks like a zoomed-out version of the **White Noise** noise type.
- **Value Fractal**: First, this algorithm generates noise in a similar way as the **Value** noise type. Then, it creates fractals using those values.
- **Perlin**: The perlin noise algorithm generates noise in a more natural, ordered sequence. This noise type is well-suited for modeling natural phenomena such as clouds, water, and vegetation distributions.
• **Perlin Fractal**: First, this algorithm generates noise in a similar way as the **Perlin** noise type. Then, it creates fractals using those values.

• **Simplex**: This algorithm generates noise in a similar way as the Perlin noise type, but with less directional artifacts and improved computational complexity.

• **Simplex Fractal**: First, this algorithm generates noise in a similar way as the **Simplex** noise type. Then, it creates fractals using those values.

• **Cellular**: First, the cellular noise algorithm selects random feature points throughout the world and assigns each a value. Then, the gradient signals mapped to the world coordinates have the same value as its closest feature point. The result looks like clusters of different values.

• **White Noise**: This algorithm generates pseudo-random values using the world coordinates. The result produces extremely different values for adjacent world points.

• **Cubic**: First, this algorithm generates noise in a similar way as the **White Noise** noise type. Then each value is interpolated with other surrounding values using a cubic function. The result looks similar to the **Perlin** noise type, but with less directional artifacts and a higher occurrence of extreme values.

• **Cubic Fractal**: First, this algorithm generates noise in a similar way as the **Cubic** noise type. Then, it creates fractals using those values.

### Random Seed

The algorithm uses this value to generate a random number sequence. Different values generate different sequences. The slider goes from 1 to 100, but you can enter any value from 1 to 214748647.

### Frequency

*A value that scales the generated noise.* A smaller value zooms into the noise, making the noise appear larger and less frequent. A larger value zooms out of the noise, making the noise appear smaller and more frequent.

### Octaves

*Fractal noise types only.* The number of repetitions in the algorithm to generate Fractal noise. A higher value causes finer details in the noise. Values greater than 4 are almost unnoticeable.

### Lacunarity

*Fractal noise types only.* A multiplier that defines how much the frequency changes with each octave. A smaller value zooms into the noise, making the noise appear larger and less frequent. A larger value zooms out of the noise, making the noise appear smaller and more frequent.

### Gain

*Fractal noise types only.* A multiplier that blends between the lower octaves and the higher octaves. A smaller blends towards the lower octaves, a higher value blends towards the higher octaves.

### Distance Function

*Cellular noise types only.* The cellular noise algorithm uses the distance function to calculate which cell values to use for a given point in the world. It affects how the cellular shape is distorted.

### Return Type

*Cellular noise types only.* The cellular noise calculation returns this value and uses it to generate the gradient signal.

- **Distance2Div** the distance of the closest feature point divided by the distance of the second-closest feature point.
- **Distance** the distance to the nearest feature point at any given world position.
- **Distance2** the distance to the second-nearest feature point at any world position.
- **Distance2Add** the distance of the two closest feature points added together.
• **Distance2Mul** the distance of the two closest feature points multiplied together.
• **Distance2Sub** the distance of the two closest feature points subtracted together.
• **CellValue** the value of the nearest feature point at any given world position.

**Jitter**

*Cellular noise types only.* A value that scales the location of the feature points. For ideal results, use a value between 0.0 and 1.0, where 0.0 provides no jitter.

**Advanced Settings**

In addition, the **FastNoise Gradient** component has the following advanced properties:

**Interpolation**

*Value, Value Fractal, Perlin, and Perlin Fractal noise only.* The noise generation algorithm uses this type of interpolation when calculating noise values.
• **Linear** interpolation provides angular artifacts.
• **Hermite** interpolation provides smoothed, blurred noise.
• **Quintic** interpolation is similar to hermite, but provides more defined edges without calculating angular artifacts.

**Fractal Type**

*Fractal noise types only.* The noise generation algorithm uses this type of fractal when calculating the noise values.
• **FBM (Fractional Brownian Motion)** adds multiples of frequencies and amplitudes of the noise signal together.
• **Billow** is a variant of FBM where the absolute value of multiple frequencies and amplitudes are added together. This produces extreme gradient signal values.

• **Rigid Multi** is also a variant of FBM, where the inverse of the absolute value of multiple frequencies and values of the noise signal are subtracted from each other. This produces extreme elevations in the gradient signal values.

## Fog Volume

This feature is in *preview* release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Fog Volume** component to create fog effects in your game. For more information, see Volumetric Fog (p. 1825).

By default, the **Fog Volume** component renders in nonvolumetric mode. To enable volumetric rendering, you must specify the `e_VolumetricFog` console variable. For more information, see Console Variables for Volumetric Fog (p. 1827) and Using the Console Window (p. 210).

**Note**

The **Fog Volume** component must have a **Box Shape** (p. 818) component attached to the entity. The entity uses the **Box Shape** component to set the area of the fog.

You can also change global fog properties in the **Time of Day** editor. For more information, see Setting Global (Time of Day) Fog (p. 1823).
Contents

- Fog Volume Properties (p. 594)
  - Rendering General (p. 595)
  - Fall Off Settings (p. 595)
  - Ramp (Volumetric Fog Only) (p. 596)
  - Density Noise (Volumetric Fog Only) (p. 597)
  - Light Component Fog Properties (p. 597)
- Volumetric Fog Examples (p. 599)

Fog Volume Properties

The Fog Volume component has the following properties:

Volume type

Specify the volume shape, which can be an ellipsoid or cuboid. A Box Shape (p. 818) component attached to the entity defines the boundaries for both shapes.

Color

Color of the fog. Specify a color with the Color Picker.

Default: 255, 255, 255

HDR Dynamic (Nonvolumetric fog only)

Specify the brightness of the fog color.

Default: 0

Valid values: 0 – 20

Use global fog color

If true, the Color property is ignored, and the global fog color is used instead. In the Time of Day editor, you can specify the global fog color with the Color (bottom) parameter. For more information, see Setting Global (Time of Day) Fog (p. 1823).

Default: False

Soft edges

Specify a factor to soften the edges of the fog volume. When you view the fog volume, the fog gradually fades into the surrounding area.

Default: 1

Valid values: 0 to 1

Wind influence (Volumetric fog only)

Specify the influence that the global wind has on the fog volume.

Default: 1

Valid values: 0 to 20

Fog Density

Specify the density of the fog volume.
Density offset
Specify the offset of the density of the fog volume. Specify higher values to reduce fog density around the edges.
Default: 0
Valid values: -1000 to 1000

Near cutoff
Stops rendering the fog volume, depending on the camera distance from the object, in meters.
Default: 0
Valid values: 0 to 2

Rendering General
The Fog Volume component has the following rendering options:

Minimum spec
Minimum specifications for the fog volume to be active.
Default: Low

View distance multiplier
Adjusts the maximum view distance.
Default: 1
Valid values: 0 to ∞ (infinity)

Ignore vis. areas
Specify whether the Fog Volume component affects visible areas and clip volumes. If true, the fog volume does not affect visible areas and clip volumes.
Default: False

Affect this area only
Specify whether the Fog Volume component affects only the specified area. If false, the Fog Volume component can affect multiple visible areas and clip volumes.
Default: False

Fall Off Settings
The Fog Volume component has the following fall off settings:

Longitude
Direction of the fall off or where the fog starts to fade in the world space. Specify a value of 0 so that the fog starts to fade on the east side of the Fog Volume component.
Fog Volume

Default: 0
Valid values: 0 to 360

**Latitude**

Direction of the fall off or where the fog starts to fade in the world space. Specify a value of 90 so that the fog starts to fade on top of the Fog Volume component.

For more information, see Volumetric Fog Examples (p. 599).

Default: 90
Valid values: 0 to 360

**Shift**

Specify how much to move the fog density along the fall-off direction. Specify a positive value to move thicker fog layers along the fall off direction of the fog volume. Specify a negative value to move thicker layers along the opposite direction.

Default: 0
Valid values: −50 — 50

**Scale**

Scales the density distribution along the fall-off direction. Specify higher values to make the fog fall off more rapidly.

Default: 1
Valid values: −50 to 50

**Ramp (Volumetric Fog Only)**

The Fog Volume component has the following ramp options:

**Start**

Specify the start distance of the fog density ramp, in meters.

Default: 1
Valid values: 0 to 30000

**End**

Specify the end distance of the fog density ramp, in meters.

Default: 50
Valid values: 0 to 30000

**Influence**

Specify the influence of the fog density ramp.

Default: 0
Valid values: 0 to 1
Density Noise (Volumetric Fog Only)

The Fog Volume component has the following density noise options:

**Scale**

Specify the noise value for the fog volume density. This parameter defines the thickness of the individual patches of fog. Specify a low value to thin out the fog density and increase the space between patches of fog.

Default: 1

Valid values: 0 to 10

**Offset**

Offsets the noise value for the density. Specify a value to break the solid shape of the fog volume into patches.

Default: 1

Valid values: –2 to 2

**Time frequency**

Specify the time frequency of the noise for the density. Specify high frequencies to create fog that changes quickly. You can use this parameter to cause the individual fog patches to morph into different shapes over the course of their lifetime.

*Note*

As a best practice, specify a low value. High values can cause the fog to morph too quickly.

Default: 0

Valid values: 0 to 1

**Spatial frequency**

Specify the spatial frequency of the noise for the density. You can use this parameter to define the amount of patchy fog in the fog volume. You can increase the z-value to create a layered effect in the volume; this stacks fog patches on top of each other. Specify higher x- and y-values to create more individual fog patches within the volume.

Default: 10, 10, 10

Light Component Fog Properties

If you add light components to your entities, you can also specify following parameters for light components:

**Volumetric Fog**

Enables the light to affect volumetric fog.

**Volumetric Fog Only**

Enables the light to affect volumetric fog and not affect other objects, such as meshes.

Example

In the following example, the parameter **Volumetric Fog On** is enabled for the light component.
Example

In the following example, the parameter Volumetric Fog is disabled for the light component.

For more information about light components, see Rendering (p. 534).
Volumetric Fog Examples

Example

In the following example, the value for **Latitude** is 0. The fall off appears on the left side of the fog volume, along the x-axis.

![Fog Volume Example 1](image1.png)

Example

In the following example, the value for the **Latitude** is 90. The fall off appears on the top for the fog volume.

![Fog Volume Example 2](image2.png)
**Note**
For the **Fog Volume** component, the **Latitude** and **Longitude** parameters work in the world space. Local transform changes do not affect these parameters. The **Latitude** and **Longitude** parameters orient the fog volume on the xy-plane. You can specify different values to orient the fog volume in any direction.

**Geom Cache**

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

The **Geometry Cache** component renders mesh data and can play vertex-based animations from **Alembic** files. With this, you can simulate highly complex geometry effects that might otherwise be unachievable.
Geom Cache Parameters

Use the following parameters to customize your geometry cache.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast Shadows</td>
<td>When selected, the geometry cache casts shadows.</td>
</tr>
<tr>
<td>Visible</td>
<td>When selected, the geometry cache is visible.</td>
</tr>
<tr>
<td>Use Vis Areas</td>
<td>When selected, the geometry cache can be obscured by vis areas and portals.</td>
</tr>
<tr>
<td>Play on Start</td>
<td>When selected, the geometry cache animation plays in the editor and plays when the component initializes.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Loop</td>
<td>When selected, the geometry cache animation loops indefinitely.</td>
</tr>
<tr>
<td>View Distance Multiplier</td>
<td>This value is multiplied by the Max View Distance parameter to get the true maximum viewing distance.</td>
</tr>
<tr>
<td>LOD Distance Ratio</td>
<td>This affects how LODs are chosen. A lower value means that less detailed LODs are used at shorter view distances.</td>
</tr>
<tr>
<td>Start Time</td>
<td>The time in seconds into the animation that the component start playing at. For example, use 0.0 to start at the beginning. Use 1.0 to start 1 second into the animation.</td>
</tr>
<tr>
<td>Min Spec</td>
<td>The minimum engine spec that this Geometry Cache renders at. To render at all specs, select Low.</td>
</tr>
<tr>
<td>Max View Distance</td>
<td>The maximum distance from which this geometry cache can be seen. This value is multiplied by the View Distance Multiplier to get the true maximum view distance.</td>
</tr>
<tr>
<td>Geometry Cache Asset</td>
<td>The geometry cache asset to render.</td>
</tr>
<tr>
<td>Stand-in</td>
<td>The entity to render (instead of the geometry cache) when the current camera is farther away than the Stand-in Distance. The given entity must have a mesh component attached.</td>
</tr>
<tr>
<td>Last Frame Stand-in</td>
<td>The entity to render (instead of the geometry cache) when the animation is on the last frame. The given entity must have a mesh component attached.</td>
</tr>
<tr>
<td>First Frame Stand-in</td>
<td>The entity to render (instead of the geometry cache) when the animation is on the first frame. The given entity must have a mesh component attached.</td>
</tr>
<tr>
<td>Stand-in Distance</td>
<td>Defines how far away the current camera must be from the geometry cache's center before the Stand-in takes its place.</td>
</tr>
<tr>
<td>Stream In Distance</td>
<td>Defines how close the camera has to be for the geometry cache to start streaming in animation data to prepare for animation.</td>
</tr>
<tr>
<td>Material Override</td>
<td>An override material to use instead of the one given by the geometry cache asset.</td>
</tr>
</tbody>
</table>
GeometryCacheComponentRequests Interface

This EBus is used to communicate to an entity with a **Geometry Cache** component. This is available at runtime and at edit time and can be accessed from C++, Lua and ScriptCanvas. Some functionality may be restricted to C++ exclusively.

For more information about using the event bus (EBus) interface, see [Working with the Event Bus (EBus) system](p. 1851).

**GeometryCacheComponentRequests**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play</td>
<td>Begins playing the geometry cache's animation. If the animation is already playing, this does nothing. From here the animation can be paused with Pause or stopped with Stop.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Pause</td>
<td>Pauses the geometry cache's animation. From here the animation can be resumed by calling Play.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops the geometry cache's animation. The animation is brought back to the First Frame where the First Frame Stand-in can be used. From here the animation can be restarted with Play.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetTimeRemaining</td>
<td>Returns how much time remains in the animation. If the animation is paused or stopped this returns -1.0.</td>
<td>None</td>
<td>float</td>
<td>Yes</td>
</tr>
<tr>
<td>GetCurrentStandinType</td>
<td>Returns which Stand-in is currently visible.</td>
<td>None</td>
<td>Stand-in type</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>---------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>Note that in Lua and Script Canvas this is returned as a number, rather than as an Enum as in C++. 0: No Stand-in 1: First Frame stand-in 2: Last Frame stand-in 3: Distance-based stand-in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetGeomCacheAsset</td>
<td>Sets the given geometry cache asset. You can also use this to dynamically change which geometry cache is rendered on the fly.</td>
<td>AZ::Data::AssetId</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>GetGeomCacheAsset</td>
<td>Gets the geometry cache asset currently in use. If no geometry cache is in use it returns an invalid asset.</td>
<td>None</td>
<td>AZ::Data::Asset<a href="">AZ::Data::AssetData</a></td>
<td></td>
</tr>
<tr>
<td>SetVisible</td>
<td>Sets whether or not the geometry cache is to be processed for rendering. If visibility is turned off, all stand-ins for this geometry cache are also turned off.</td>
<td>bool</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>GetVisible</td>
<td>Gets whether or not the geometry cache is to be processed for rendering. If visibility is turned off, all stand-ins for this geometry cache are also turned off. Note that if this returns true, that doesn't mean that the geometry cache is currently in view, just that it will be processed by the rendering engine. The geometry cache can still be out of view and not be currently on screen.</td>
<td>None</td>
<td>bool</td>
<td>Yes</td>
</tr>
<tr>
<td>SetLoop</td>
<td>Sets whether or not the geometry cache animation loops. The Last Frame Stand-in will never be visible as long as this remains true.</td>
<td>bool</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetLoop</td>
<td>Gets whether or not the geometry cache animation is set to loop.</td>
<td>None</td>
<td>bool</td>
<td>Yes</td>
</tr>
<tr>
<td>SetStartTime</td>
<td>Sets the time point (in seconds) that the geometry cache animation should start at. Changing the start time of the animation restarts the animation.</td>
<td>float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetStartTime</td>
<td>Gets the current start time point (in seconds) for the geometry cache animation.</td>
<td>None</td>
<td>float</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>SetStreamInDistance</td>
<td>Sets the distance threshold that controls geometry cache streaming. When the distance between the center of the geometry cache and the current camera is greater than this value, the geometry cache's animation begins to stream into memory.</td>
<td>float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetStreamInDistance</td>
<td>Gets the distance threshold that controls when the geometry cache streams to memory.</td>
<td>None</td>
<td>float</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Geom Cache

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetFirstFrameStandIn</td>
<td>Sets the entity to be used for the First Frame Stand-in.</td>
<td>AZ::EntityId</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>It's assumed that the entity id points to an entity that has a mesh component attached. The stand-in is controlled by the visibility parameter of the mesh component. In the editor interface, setting a first frame stand-in automatically parents the new stand-in and un-parents the old stand-in; this behavior does not exist on these bus calls. Invalid entity ids are ignored.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This stand-in is used until the geometry cache animation starts playing.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetFirstFrameStandIn</td>
<td>Gets the entity that is used as the First Frame Stand-in.</td>
<td>None</td>
<td>AZ::EntityId</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>SetLastFrameStandIn</td>
<td>Sets the entity to be used for the Last Frame Stand-in.</td>
<td>AZ::EntityId</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>It's assumed that the entity id points to an entity that has a mesh component attached. The stand-in is controlled by the visibility parameter of the mesh component. In the editor interface, setting a last frame stand-in automatically parents the new stand-in and unparents the old stand-in; this behavior does not exist on these bus calls. Invalid entity ids are ignored. This entity is never used as long as the Loop parameter remains true. This stand-in is not used until the geometry cache animation ends.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetLastFrameStandIn</td>
<td>Gets the entity that is used as the Last Frame Stand-in.</td>
<td>None</td>
<td>AZ::EntityId</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>SetStandIn</td>
<td>Sets the entity to be used for the distance-based stand-in.</td>
<td>AZ::EntityId</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>It's assumed that the entity id points to an entity that has a mesh component attached. The stand-in is controlled by the visibility parameter of the mesh component. In the editor interface, setting a stand-in automatically parents the new stand-in and un-parents the old stand-in; this behavior does not exist on these bus calls. Invalid entity ids are ignored. This Stand-in is used as long as the distance between the geometry cache's center and the current camera's position is larger than the Stand-in Distance parameter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetStandIn</td>
<td>Gets the entity that is used as the distance-based Stand-in.</td>
<td>None</td>
<td>AZ::EntityId</td>
<td>Yes</td>
</tr>
<tr>
<td>SetStandInDistance</td>
<td>Sets the distance threshold that controls the visibility of the stand-in.</td>
<td>float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>GetStandInDistance</td>
<td>Gets the distance threshold that controls the visibility of the stand-in.</td>
<td>None</td>
<td>float</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### GeometryCacheComponentNotifications Interface

This EBus is used to react to events that take place in the **Geometry Cache** component.

For more information about using the event bus (EBus) interface, see *Working with the Event Bus (EBus) system* (p. 1851).

#### GeometryCacheComponentNotifications

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnPlaybackStart</td>
<td>Event that triggers when geometry cache playback starts.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>OnPlaybackPause</td>
<td>Event that triggers when geometry cache playback pauses.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>OnPlaybackStop</td>
<td>Event that triggers when geometry cache playback stops.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>OnStandinChanged</td>
<td>Event that triggers when the geometry cache changes which stand-in is in use.</td>
<td>StandinType</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This does trigger if a stand-in is turned off and the Geometry Cache becomes active instead.

The parameter includes which StandinType the Geometry Cache has changed to.

The following is a notification bus sample script.

```plaintext
function example:OnActivate()
```
The following is a more complete example script of how to use the geometry cache from Lua. See the inline comments for descriptions of what the script is performing at that point.

```lua
-- Load helpers for handling input events from Lua.
local inputMultiHandler = require('Scripts.Utils.Components.InputUtils')

local geomcachetest = 
{ Properties = 

|-- Specify an existing entity to swap with an existing stand-in
replacementStandin = {default = EntityId()} 
},

function geomcachetest:OnActivate() 

|-- Subscribe to geometry cache notifications on this entity
self.geometryCacheNotificationHandler = GeomCacheComponentNotificationBus.Connect(self, self.entityId); 

|-- Subscribe to some input events
|-- Here it is assumed that 4 events are hooked up to input
self.inputHandlers = inputMultiHandler.ConnectMultiHandlers{
    [InputEventNotificationId("PlayAnimation")] = {
        OnPressed = function(floatValue) self:PlayAnimation(floatValue) end,
    },
    [InputEventNotificationId("PauseAnimation")] = {
        OnPressed = function(floatValue) self:PauseAnimation(floatValue) end,
    },
    [InputEventNotificationId("StopAnimation")] = {
        OnPressed = function(floatValue) self:StopAnimation(floatValue) end,
    },
    [InputEventNotificationId("ChangeStandin")] = {
        OnPressed = function(floatValue) self:ChangeStandin(floatValue) end,
    },
},

|-- An identifier for the stand-in that is currently in use
|-- 0 indicates no stand-in. See :OnStandinChanged
self.currentStandin = 0;

end

function geomcachetest:OnDeactivate() 

self.geometryCacheNotificationHandler:Disconnect();

end

-- These events trigger when input is selected
|-- The first three are simple playback controls. Last one swaps out the stand-in mesh
function geomcachetest:PlayAnimation(value)
    GeometryCacheComponentRequestBus.Event.Play(self.entityId);
end

function geomcachetest:PauseAnimation(value)
    GeometryCacheComponentRequestBus.Event.Pause(self.entityId);
end

function geomcachetest:StopAnimation(value)
    GeometryCacheComponentRequestBus.Event.Stop(self.entityId);
end

function geomcachetest:ChangeStandin(value)
    self.replacementStandin = value;
end

function geomcachetest:OnStandinChanged(value)
    print("Standin changed to " .. value);
end
```

---

**Geom Cache**

Lumberyard User Guide

---

Version 1.28

611
function geomcachetest:PauseAnimation(value)
    GeometryCacheComponentRequestBus.Event.Pause(self.entityId);
end

function geomcachetest:StopAnimation(value)
    GeometryCacheComponentRequestBus.Event.Stop(self.entityId);
end

function geomcachetest:ChangeStandin(value)
    GeometryCacheComponentRequestBus.Event.SetStandIn(self.entityId,
            self.Properties.replacementStandin);
end

--These are the geometry cache notifications
--Emits a simple debug log when events happen

function geomcachetest:OnPlaybackStart()
    Debug.Log("GeometryCache: Playback Started");
end

function geomcachetest:OnPlaybackPause()
    Debug.Log("GeometryCache: Playback Paused");
end

function geomcachetest:OnPlaybackStop()
    Debug.Log("GeometryCache: Playback Stopped");
end

function geomcachetest:OnStandinChanged(standinType)
    Debug.Log(string.format("GeometryCache: Changed standin from %d to %d",
            self.currentStandin, standinType));
    self.currentStandin = standinType;
end

return geomcachetest

Altitude Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

Constant Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

Image Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.
Random Noise Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Reference Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Slope Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Shape Falloff Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Surface Mask Gradient

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Perlin Noise Gradient

Dither Gradient Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Invert Gradient Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Levels Gradient Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Gradient Mixer

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Posterize Gradient Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Smooth-Step Gradient Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Threshold Gradient Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Gradient Transform Modifier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

High Quality Shadow

This feature is in preview release and is subject to change.

Version 1.28
614
You can use the **High Quality Shadow** component to give an entity its own shadow map that is a higher resolution than you could achieve with a global shadow map. This component applies to light from the sun and helps provide crisp, well-defined shadows. You can use this on main characters or prominent elements to improve their appearance.

This component can impact performance and memory, and should be used sparingly. As such, it can be useful to dynamically change the `Enabled` setting as performance needs change during the game. Use the `Set Enabled (p. 2670)` node to enable or disable the **High Quality Shadow** component in the Script Canvas editor. For example, if your main character has a high quality shadow, you can enable it during cutscenes and disable it during gameplay.

For more information, see **Shadow Nodes (p. 2669)**.

### High Quality Shadow Component Properties

The **High Quality Shadow** component has the following properties:

- **Enabled**
  
  Enable the shadow map.

- **Const Bias**

  Avoid the self shadow artifacts.

- **Slope Bias**

  Avoid the self shadow artifacts.

- **Jitter**

  Filter kernel size, which directly affects shadow softness.

- **Bounding Box Scale**

  Set the scale factor for the bounding box of the selected entity. This is useful if the engine bounding box is too small or too large.

- **Shadow Map Size**

  Set the size of the custom shadow map. The value automatically rounds to the next power of two.

### Infinite Ocean

You can use the **Infinite Ocean** component to add an ocean surface to your level. The ocean expands to the horizon, no matter where the camera moves in the environment.

- Ocean simulation is based on a fast Fourier transform (FFT) noise-based gradient, normal, and height maps, and mixes multiple frequencies and scale for ambient waves.
• Players can interact with ocean waves and generate wave propagation, such as shooting at the water.
• On higher-end machines, vertex displacement is used and supports additional water interaction, such as buoyancy and floating objects.

Note
To enable the Infinite Ocean component, you must enable the Water gem. For more information, see Add modular features and assets with Gems (p. 1064).

Example Infinite Ocean Component

Water rendering in real time is complex due to visual and physical contributing factors. For example, light behaves differently when it moves from air into water. Most sunlight reflects off the ocean, but a large amount also penetrates into the water, which absorbs light more strongly than air. The visible light consists of a rainbow with different wavelengths, which appears as different colors.

For example, blue light penetrates farther in the water, giving the ocean its blue and green colors. In contrast, the ocean absorbs red, orange, and yellow wavelengths, which removes these colors. When light hits a substance, the light does one of the following:

• The light hits molecules of a substance and then scatters and bounces off in different directions.
• The light passes through the substance.
• The substance absorbs the light, either completely or only some wavelengths. In Lumberyard this is called underwater fog.
In Lumberyard, various elements such as sunlight, reflections, and environment probes contribute to how the ocean surface renders. These elements approximate how light is scattered and absorbed in the ocean.

**Topics**
- Adding the Infinite Ocean Component (p. 617)
- Understanding Water Properties (p. 617)
- Infinite Ocean Component Properties (p. 618)
- Water Shader Material for the Infinite Ocean Component (p. 629)
- Best Practices Working with the Infinite Ocean Component (p. 630)
- Additional Workflow (p. 630)
- Console Variables for the Infinite Ocean Component (p. 632)
- Infinite Ocean Component with the Track View Editor (p. 634)
- Infinite Ocean Component Request Bus Interface (p. 635)
- Using the Infinite Ocean Component with Script Canvas (p. 641)

**Adding the Infinite Ocean Component**

**To create an infinite ocean in your level**

1. Create an entity (p. 463) in your level.
2. In the **Entity Inspector**, click **Add Component**, and then choose the **Infinite Ocean** component.

   For more information, see Working with Components (p. 479).

**Understanding Water Properties**

When working with water properties, the color and appearance come from two places:

- Reflections that occur on the surface.
- How light interacts with water molecules and suspended particles in the water. In Lumberyard this is called *underwater fog*.

The following images demonstrate how bright highlights come from the sky reflection, mostly at sharp angles or in bright reflection points. Notice how the underwater fog appears differently, as particles such as algae, dust, and plankton can change the appearance of the water.

**Example**

In the following example, the underwater fog appears greenish.
Example

In the following example, the underwater fog appears brown.

Avoid using only the reflection to achieve the appearance that you want. The water appearance is the sum of the reflection on the ocean surface and the underwater fog below the surface.

Infinite Ocean Component Properties

Use the following parameters for the Infinite Ocean component to change how the ocean appears.

Contents
- General (p. 619)
- Animation (p. 619)
- Fog (p. 620)
- Caustic (p. 622)
- Reflection (p. 623)
- Advanced (p. 624)
- Tessellation Examples (p. 624)
General

See the following general parameters.

Enable Ocean Bottom

Toggle the infinite plane below the ocean.

Default value: True

Water Material

Render the ocean with this assigned material. The material must be a water shader.

You can find the default material in the \lumberyard_version\dev\Engine\EngineAssets\Materials\Water\ocean_default.mtl directory.

Default value: ocean_default.mtl

Animation

Use the following animation parameters to control the movement of the waves and the scrolling of the water texture, with the wind speed and direction.

Waves Amount

Specify the frequency of ocean waves. Higher values create waves that appear smaller and closer together.

Default value: 0.75

Valid values: 0.20 to 5.0

Waves Size

Specify the height of ocean waves.

Tip

For Wave Size, higher values work better with lower values for Waves Amount.

Default value: 1.25

Valid values: 0 to 3.0

Waves Speed

Specify the speed of ocean wave movement.

Default value: 1.0

Valid values: 0 to 5.0

Wind Direction

Specify the direction of the ocean wind, in radians.

Default value: 1.0

Valid values: 0 to 6.28

Wind Speed

Specify the speed of the ocean wind.
Fog

Use the following fog parameters to control the density and color of the ocean (underwater fog). This simulates how light bounces off, is absorbed by, and scatters with the particulate matter in the water.

Color

Specify the color to render the underwater fog.

Default value: 5, 36, 32

Color Multiplier

Influence the intensity of the underwater fog color with this multiplier.

Default value: 0.15

Valid values: 0 to 1.0

Density

Specify the density of the underwater fog. Higher values make transparency fall more quickly.

Default value: 0.07

Valid values: 0 to 1.0

Near Fog Color

Reduce the underwater fog color to this color near shallow depths.

Tip

Keep the default value or specify a near-black color.

Default value: 1, 7, 5

Example Low Density

In the following example, the value for the Density parameter is low. This makes the water more transparent, so that you can see the legs of the pier and the sand.
Example High Density

In the following example, the value for the **Density** parameter is high. This makes the water appear cloudier and obscures details like the legs of the pier and sand.
Caustic

The following caustic parameters control how the refracted rays of light animate and project on objects below the ocean surface.

Enable Caustics

Apply the caustic effect of the ocean on geometry below the ocean surface.

Default value: True

Depth

Maximum depth below the ocean height at which caustic effects still appear, in meters.

Default value: 8

Valid values: 0 to 100.0

Intensity

Specify the intensity of the light in the ocean caustic effect. A value of 0 turns off caustic effects. Near-zero values show caustic effects strongly. For example, a value of 0 to 0.01 is a sudden change.

Default value: 1

Valid values: 0 to 10.0

Tiling

Specify the amount of ocean caustic effect that is tiled. Lower values spread out the effect and higher values tighten the detail.

Default value: 2
Valid values: 0.10 to 10.0

Distance Attenuation

Specify the attenuation distance (from the camera) that caustic effects of the ocean are applied.

Lower values apply caustic effects closer to the camera.

Larger values apply caustic effects farther from the camera.

Default value: 10.

Valid values: 0 to 100.0

Example Caustics

The following is an example of how light interacts with the water below the surface.

Reflection

The following reflection parameters are set for the resolution of the reflection texture map (this is based on screen size * resolution scale) and a set of flags that the reflection system uses when the camera is above the ocean.

Resolution Scale

Scale of the screen resolution that is used to render ocean reflections.

Default value: 0.50

Valid values: 0 to 1.0
Reflect Entities
Specify whether entities are included when reflections are rendered in the ocean.
Default value: True

Reflect Objects
Specify whether entities with the Mesh component attached are included when reflections are rendered in the ocean.
Default value: True

Reflect Terrain
Specify whether terrain detail materials are included when reflections are rendered in the ocean.
Default value: True

Reflect Particles
Specify whether particles are included when reflections are rendered in the ocean. For example, if your level has a flame particle effect on a torch, a value of false means that the torch flame does not reflect into the ocean.
Default value: True

Advanced
In most cases, you don't need to change the values for the advanced parameters. For example, you should set the Tessellation parameter once and not during runtime updates.

Tessellation
Specify the amount of ocean water surface geometry tessellation. For more information, see Tessellation Examples (p. 624).

Note
The Tessellation parameter affects the 3D mesh for the ocean triangles. Higher values create more polygons and a smoother ocean surface, at a higher performance cost.

Default value: 85
Valid values: 10 to 500

Godrays Enabled
Enable god rays (or sun beams) to appear below the ocean surface.
Default value: True

Underwater Distortion
Specify the amount of the scene that is distorted when it renders while the camera is underwater. For best practice, specify a value less than 1.0.

Default value: 1
Valid values: 0 to ∞

Tessellation Examples
The Infinite Ocean component has controls for setting the level of tessellation (number of vertices and polygons) for the generated mesh that defines the surface of the ocean.
Note
We recommend that you set and configure the **Tessellation** parameter once to achieve the desired look and feel for your level. Although you can change this value at run time, this forces Lumberyard to rebuild the mesh, which can cause an undesirable hitch in performance. If you must change the tessellation value during run time, we recommend that you do so under controlled conditions where you can hide the changes.

**Example Tessellation = 10, low setting**

At lower settings, there is a noticeable impact on the visual quality.
In the following example, the lower visual quality is more noticeable with higher values for **Waves Size** parameter. There are fewer vertices to displace in the simulation, so the waves are not well-defined.

Example Tessellation = 85, default setting

In most cases, the default settings for the **Tessellation** parameter are sufficient.
Specifying lower values for the **Waves Amount** parameter works better with higher values on the **Waves Size** parameter, but the higher tessellation smooths out the ocean surface.
Example Tessellation = 500, high setting

In the following example, higher values for the **Tessellation** parameter minimally impact the visual quality.

Specifying a high value for the **Waves Amount** parameter creates a choppier ocean surface, so it's helpful to increase the value for the **Tessellation** parameter.
Water Shader Material for the Infinite Ocean Component

The Infinite Ocean component is compatible only with a material that uses the water shader, such as the default ocean_default.mtl file. The water shader is a dedicated shader that Lumberyard uses only to render the ocean, and adds effects such as reflection, ripple, and foam.

Example Water Shader

The following is an example of how you can configure the water shader material in the Material Editor.
Note
When you set up water materials for the ocean, disable the Water Volume parameter. When you set up water material for other types of water volumes, such as lakes and rivers, enable the Water Volume parameter.

For more information, see Water Shader (p. 1715).

For lakes, rivers, and other bodies of water, use the WaterVolume Shader (p. 1719).

Best Practices Working with the Infinite Ocean Component

See the following best practices when working with the Infinite Ocean component.

• You can use the default parameters for the Infinite Ocean component and the ocean_default.mtl material to create a default ocean, but you can change the settings as needed.

  Note
  Changing the ocean_default.mtl material affects any level that has an ocean that uses the material.

• You can change the colors that contribute to how the ocean appears.
  • For the Infinite Ocean component, use the Color and Near Fog Color parameters. For more information, see Fog (p. 620).
  • For the water material (ocean_default.mtl), use the Diffuse Color and Specular Color parameters. For more information, see Water Shader Material for the Infinite Ocean Component (p. 629).

• Use both reflections and the underwater fog to achieve the water appearance that you want. For more information, see Reflection (p. 623) and Fog (p. 620).

• If your environment looks dull, you'll get dull reflections in the water. Create an interesting sky look and skyline gradient. If you are creating a night scene, use bright spots such as city lights.

• Use foam effects sparingly.

• Ocean reflection draw distance is tied to the terrain detail draw distance.

To customize the ocean appearance for a specific level, do the following:

1. Navigate to the lumberyard_version/dev/Engine/EngineAssets/Materials/Water/ocean_default.mtl directory and copy the ocean_default.mtl file to create another file.

2. Specify the new file for the Infinite Ocean component. See General (p. 619).

3. In the Material Editor, customize the material settings for your current level.

Tip
For best performance, we recommend the following:

• If your level doesn't require sun reflection, disable these parameters.

• If your level doesn't require foam, disable this parameter.

• If you don't need caustic rendering, set the Caustic Multiplier parameter to 0 to disable it.

Additional Workflow

See the following when working with the Infinite Ocean component.

Note
The e_VolumetricFog console variable is currently not compatible with the Infinite Ocean component and can cause rendering issues for underwater fog.
Slices

- You can create a slice that contains an entity with an Infinite Ocean component; however, not all slice features are supported at this time.
- You can dynamically spawn a slice that contains an Infinite Ocean component; however, asynchronous loading of the Infinite Ocean component is not supported at this time.
- Only one Infinite Ocean component can be active at a time. If you create a second Infinite Ocean component in your level or place a slice that contains another Infinite Ocean component, an error appears in the console.
- You can load more than one Infinite Ocean component in a scene, but you must manage which entity has the active component.

For example, in the Entity Inspector, use the Start Active parameter to toggle the active state of the entity to which the Infinite Ocean component is attached.

This allows you to have multiple Infinite Ocean components, each with a different look and feel, set their starting states (activate one and disable the others), and then toggle the active state during run time to control which ocean is rendered.

Parent and Child Entities

- You can use the entity's Transform (p. 878) component to control the surface height of the Infinite Ocean component.
- Specify the z-axis of the Transform component to set the Infinite Ocean component's height.
- The Infinite Ocean component respects the hierarchical transforms, so if the entity is a child nested within a hierarchy, the parent transform above it contributes to the height of the Infinite Ocean component.

Ocean Bottom

The ocean bottom feature toggles on an infinite plane that draws below the ocean. The plane draws black, and its main purpose is to fix the depth buffer behind the water in areas beyond the level geometry.

- If you are using the ocean bottom, the parent transform of the Infinite Ocean component sets the ocean’s height. If the height of the ocean is near or below 0 and you are using the ocean bottom, there can be some visual anomalies.
- Normally, the ocean bottom height is set to 0 (z-axis is 0). This setting is not controlled by the parent transform.
- Since the ocean level can be freely transformed, including negative heights below 0, the ocean bottom is configured to always draw at least 5 meters below the height of the ocean surface.

Note
Currently, you can change the ocean bottom only in the code.

To change the ocean bottom settings

1. Navigate to the lumberyard_version/dev/Engine/Shaders/HWScripts/CryFX directory.
2. For the WaterOceanBottom.cfg shader, specify a value for the MIN_WATER_DEPTH parameter.

If you want a better looking ocean bottom near the visible playable area, we recommend that you create a large bottom ocean geometry with a material assignment to match the look and feel of your level.
### Console Variables for the Infinite Ocean Component

You can specify the following console variables to enable advanced settings for the **Infinite Ocean** component. In general, you don’t need to change these settings. For more information, see Using the Console Window (p. 210).

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| e_HwOcclusionCullingWater | Specify the ocean occlusion culling so that the ocean doesn't render if it’s not visible. We recommend that you keep the default value of 1.  
0 = Off  
1 = On |
| e_WaterOcean | Enable or disable the ocean.  
0 = Disables the ocean.  
1 = Enables the ocean as normal.  
2 = Enables the ocean and fog in one frame. This can cause a small increase in performance but provides a slightly different look. |
| e_WaterOceanFFT | Enable or disable using fast Fourier transforms with water. This console variable has interdependencies with other console variables and should only be used with `sys_spec_water`.  
0 = Off  
1 = On |
| e_WaterTessellationSwathWidth | Used only in older style oceans (non-FFT water). If the value is negative, the triangle mesh generates normally. If the value is positive, the value determines the length of triangle strips used in the mesh. |
| q_ShaderWater | Control the shader quality of the ocean. This value should only be changed with other console variables that are controlled by the `sys_spec_water` console variable.  
Valid values: 0 to 3 |
| r_WaterReflectionsMinVisiblePixelsUpdate | Percentage of water reflecting pixels that are required to update the reflection every frame.  
For example, if the value is set to 0.4, the water’s reflection updates every frame as long as the ocean covers 40% or more of the screen. At least 40% of pixels is needed to show the reflection in the ocean.  
If the ocean covers less than 40% of the screen, the reflection no longer updates every frame.
<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame. The update rate is then controlled by <code>r_WaterReflectionsMinVisUpdateDistanceMul</code> and <code>r_WaterReflectionsMinUpdateFactorMul</code>.</td>
<td>0 to 1</td>
</tr>
<tr>
<td><code>r_WaterReflectionsMinVisUpdateDistanceMul</code></td>
<td>Use this multiplier with the <code>r_WaterUpdateDistance</code> console variable to determine the update rate of the water reflection. This distance is from the camera's position, the last time the reflection is rendered to the current camera position.</td>
</tr>
<tr>
<td><code>r_WaterReflectionsMinUpdateFactorMul</code></td>
<td>A multiplier on the distance the camera is from the water (this value is always 0.3 or less). This distance is based on the camera's position and is used to determine the update rate of the water reflection. This distance is not based on what is in the camera's view.</td>
</tr>
<tr>
<td><code>r_WaterTessellationHW</code></td>
<td>Enable or disable DX11-style tessellated water. This value should only be changed with other console variables that are controlled by the <code>sys_spec_water</code> console variable.</td>
</tr>
<tr>
<td><code>r_WaterUpdateDistance</code></td>
<td>Distance that the camera normally needs to move before the reflection is updated. You can change the distance with the <code>r_WaterReflectionsMinVisUpdateDistanceMul</code> console variable.</td>
</tr>
<tr>
<td><code>r_WaterUpdateFactor</code></td>
<td>Determine how long to wait between updating water reflections.</td>
</tr>
</tbody>
</table>
**sys_spec_water**

This is a special data-driven console variable that controls other console variables related to the **Infinite Ocean** component and water volumes.

You can find the console variables that are defined by this value in the `sys_spec_water.cfg` file, located in the `lumberyard_version/dev/Engine/Config/CVarGroups/` directory.

In general, you don't need to change this file, and many of the console variables that are set in this file for different system specs do not affect the **Infinite Ocean** component.

---

### Infinite Ocean Component with the Track View Editor

You can add the **Infinite Ocean** component to an entity sequence in the Track View. This lets you control the component's parameters to be keyframes and changed along the timeline of the sequence.
Note
Some Infinite Ocean component parameters are not available in the Track View. The Wind Speed and Wind Direction parameters are not supported at this time.

For more information, see Using the Track View Editor (p. 1584).

Infinite Ocean Component Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

Lua Scripting

In general, the Lua scripting API for the Infinite Ocean component maps directly to the component properties with the Get* and Set* functions. However, there are some exceptions where there is only a Get function, and you can't directly set a value, such as the ocean height.

Note
You can specify float numbers for most functions, but some require an enumeration and/or a boolean value.

Example

```lua
function example:OnActivate()
    OceanEnvironmentRequestBus.Broadcast.SetFogDensity(2)
end
```

OceanLevel

Returns the height of the ocean in the world z-axis.

Note
There is no function to set the ocean level, since this parameter is attached to the Transform component.

Parameters

None

Return

Type: Float

Scriptable

Yes

Example

```lua
```

WaterTessellationAmount

Sets the amount of geometry tessellation for rendering the ocean surface.
Parameters
Type: Int

Return
Type: Int

Scriptable
Yes

Example
OceanEnvironmentRequestBus.Broadcast.SetWaterTessellationAmount(int)
OceanEnvironmentRequestBus.Broadcast.GetWaterTessellationAmount()

AnimationWindDirection and AnimationWindSpeed
Sets the direction and speed of the ocean wind.

Parameters
Type: Float

Return
Type: Float

Scriptable
Yes

Example
OceanEnvironmentRequestBus.Broadcast.SetAnimationWindDirection(float)
OceanEnvironmentRequestBus.Broadcast.GetAnimationWindDirection()
OceanEnvironmentRequestBus.Broadcast.SetAnimationWindSpeed(float)
OceanEnvironmentRequestBus.Broadcast.GetAnimationWindSpeed()

AnimationWavesSpeed, AnimationWavesSize, and AnimationWavesAmount
Specifies the animation parameters of the ocean waves, such as speed, size, and amount.

Parameters
Type: Float

Return
Type: Float

Scriptable
Yes

Example
OceanEnvironmentRequestBus.Broadcast.SetAnimationWavesSpeed(float)
OceanEnvironmentRequestBus.Broadcast.GetAnimationWavesSpeed()
OceanEnvironmentRequestBus.Broadcast.SetAnimationWavesSize(float)
OceanEnvironmentRequestBus.Broadcast.GetAnimationWavesSize()
OceanEnvironmentRequestBus.Broadcast.SetAnimationWavesAmount(float)
**ReflectResolutionScale**

Specifies the scale of the screen resolution to use for rendering ocean reflections.

**Parameters**
- Type: Float

**Return**
- Type: Float

**Scriptable**
- Yes

**Example**

```csharp
OceanEnvironmentRequestBus.Broadcast.SetReflectResolutionScale(float)
OceanEnvironmentRequestBus.Broadcast.GetReflectResolutionScale()
```

**ReflectionAnisotropic**

Enables the anisotropic filter for rendered ocean reflections.

**Parameters**
- Type: Boolean

**Return**
- Type: Boolean

**Scriptable**
- Yes

**Example**

```csharp
OceanEnvironmentRequestBus.Broadcast.SetReflectionAnisotropic(boolean)
OceanEnvironmentRequestBus.Broadcast.GetReflectionAnisotropic()
```

**ReflectRenderFlag**

For each reflection flag, you can specify how the ocean renders reflections.

You can specify the following flags:

- `ReflectionFlags.TerrainDetailMaterials`
- `ReflectionFlags.StaticObjects`
- `ReflectionFlags.Particles`
- `ReflectionFlags.Entities`

**Parameters**
- Type: Enum flag and boolean
Return

Type: Boolean
Scriptable
Yes

Example

OceanEnvironmentRequestBus.Broadcast.SetReflectRenderFlag(ReflectionFlags.TerrainDetailMaterials, boolean)

UseOceanBottom

Toggles the infinite plane below the ocean.

Parameters

Type: Boolean

Return

Type: Boolean
Scriptable
Yes

Example

OceanEnvironmentRequestBus.Broadcast.SetUseOceanBottom(boolean)
OceanEnvironmentRequestBus.Broadcast.GetUseOceanBottom()

GodRaysEnabled

Enables god rays (sun beams) under the ocean.

Parameters

Type: Boolean

Return

Type: Boolean
Scriptable
Yes

Example

OceanEnvironmentRequestBus.Broadcast.SetGodRaysEnabled(boolean)

UnderwaterDistortion

Specifies the amount of the scene that is distorted when it renders while the camera is underwater.
Parameters
Type: Float
Return
Type: Float
Scriptable
Yes
Example
OceanEnvironmentRequestBus.Broadcast.SetUnderwaterDistortion(float)

FogColor, FogColorMultiplier, and NearFogColor

Specifies the color for the underwater fog of the ocean.

You can specify a color value, such as Color(1.0,2.0,3.0,1.0), and then pass it as an argument like the following: SetFogColor(Color(1.0,2.0,3.0,1.0))

The FogColorPremultiplied function is applied to the ocean's fog color. You can use the GetFogColorPremultiplied function to return the result of Fog Color * Fog Color Multiplier.

Note
Currently, you cannot directly specify values for the FogColorPremultiplied function.

For SetFogColorMultiplier and GetFogColorMultiplier functions:
Parameters
Type: Float
Return
Type: Float
Scriptable
Yes

For SetFogColor, GetFogColor, SetNearFogColor, and GetNearFogColor functions:
Parameters
Type: Color
Return
Type: Color
Scriptable
Yes

Example
OceanEnvironmentRequestBus.Broadcast.SetFogColor(Color)
OceanEnvironmentRequestBus.Broadcast.GetFogColor()
CausticsEnabled

Applies the caustics effect of the ocean on geometry below the ocean surface.

**Parameters**
- Type: Boolean

**Return**
- Type: Boolean

**Scriptable**
- Yes

**Example**

```csharp
OceanEnvironmentRequestBus.Broadcast.SetCausticsEnabled(boolean)
OceanEnvironmentRequestBus.Broadcast.GetCausticsEnabled()
```

CausticsDepth, CausticsIntensity, CausticsTiling, CausticsDistanceAttenuation

Specifies the caustics parameters for the Infinite Ocean component.

**Parameters**
- Type: Float

**Return**
- Type: Float

**Scriptable**
- Yes

**Example**

```csharp
OceanEnvironmentRequestBus.Broadcast.SetCausticsDepth(float)
OceanEnvironmentRequestBus.Broadcast.GetCausticsDepth()
OceanEnvironmentRequestBus.Broadcast.SetCausticsIntensity(float)
OceanEnvironmentRequestBus.Broadcast.GetCausticsIntensity()
OceanEnvironmentRequestBus.Broadcast.SetCausticsTiling(float)
OceanEnvironmentRequestBus.Broadcast.GetCausticsTiling()
OceanEnvironmentRequestBus.Broadcast.SetCausticsDistanceAttenuation(float)
OceanEnvironmentRequestBus.Broadcast.GetCausticsDistanceAttenuation()
```

OceanMaterialName

Renders the ocean with the assigned material.

You can find the default material in the `lumberyard_version/dev/Engine/EngineAssets/Materials/Water/ocean_default.mtl` directory.

**Parameters**
- The path to the ocean material file.
Infinite Ocean

Type: String
Return

Type: String

Scriptable
Yes

Example

OceanEnvironmentRequestBus.Broadcast.SetOceanMaterialName(string)
OceanEnvironmentRequestBus.Broadcast.GetOceanMaterialName()

Using the Infinite Ocean Component with Script Canvas

You can use the Infinite Ocean component with the Script Canvas editor to change the component parameters at run time, using a visual scripting workflow. In the following example, the Script Canvas script changes the size of the ocean waves over time.

To use the Infinite Ocean component with the Script Canvas editor

1. In Lumberyard Editor, create an entity (p. 463) and attach the Infinite Ocean component. Ensure that the component is positioned correctly in your level.
2. In the Entity Inspector, for Wave Size, enter 0. This creates a flat ocean, so you can see how the Script Canvas editor interacts with the component.
3. Click Add Component and then click the Script Canvas (p. 818) component.
4. In the Script Canvas component, click the icon to open the Script Canvas editor.
5. Select the entity with the Infinite Ocean component attached and drag it to the Script Canvas editor.
6. In the Script Canvas editor, save the file as oceanscript.scriptcanvas.
7. In the Node Palette, search for the On Entity Activated node and drag it to the canvas.
   • In the canvas, from the Ocean EntityRef node, drag Get to connect it to the Source pin of the Game Entity node.
8. In the Node Palette, search for the Timer node and drag it to the canvas.
   • In the canvas, from the Game Entity node, drag the Out pin to connect it to the Start pin of the Timer node.
9. In the Node Palette, search for the Sin node and drag it to the canvas. This node provides a value that goes up and down.
   a. In the canvas, from the Timer node, drag the Out pin to connect it to the In pin of the Sin node.
   b. Drag the Seconds pin to connect it to the Angle pin of the Sin node.
   Note
   The Sin node returns values that are –/+. Negative numbers cause errors with the Wave Size parameter. Use the Add node to ensure that only positive values are returned.
10. In the Node Palette, search for the Add node and drag it to the canvas.
    a. In the canvas, from the Sin node, drag the Out pin to connect it to the In pin of the Add node.
    b. Drag the Angle pin and to connect it to the Value A pin of the Add node.
    c. For the Add node, enter 1.0 for Value B. This shifts the sine wave from (-1.0 to 1.0) to (0.00 to 2.00).
11. In the Node Palette, search for the SetAnimationWavesSize node and drag it to the canvas.
a. In the canvas, from the Add node, drag the Out pin to connect it to the In pin of the SetAnimationWavesSize node.
b. Drag the Result pin to connect it to the Number:0 pin of the SetAnimationWavesSize node.

Your script should look like the following:

![Image of script canvas]

12. Save your script.
13. In the Entity Inspector, select the entity with the Script Canvas component attached. Click the browse (...) button and navigate to the oceanscript.scriptcanvas file. Select oceanscript.scriptcanvas and then click OK.
14. To start the game, press Ctrl+G. You can see the size of the ocean waves increase and decrease over time.

**Input**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Input component to bind raw input to events in your game. The Input component references an .inputbindings file, which binds a set of inputs (such as from a mouse, keyboard, game controller, and so on) to an event.

For example, you can add the Input component to an entity and specify in the .inputbindings file that when you press the keyboard spacebar, the entity rotates.

**Note**

To work with inputs, you must enable the Input Management Framework (p. 1150) and the Starting Point Input (p. 1206) gems. For more information, see Add modular features and assets with Gems (p. 1064).

**Topics**

- Input Properties (p. 642)
- Working with the Input Component (p. 643)
- Input Component EBus Interface (p. 648)

**Input Properties**

![Image of Input properties]
The **Input** component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input to event bindings</strong></td>
<td>References an <code>.inputbindings</code> file that defines the bindings of raw input to events.</td>
</tr>
<tr>
<td><strong>Input contexts</strong></td>
<td>A string to specify valid context(s) for the <code>.inputbindings</code> file.</td>
</tr>
</tbody>
</table>

An empty string "" is the default context. This context is active whenever it's explicitly pushed to the top of the input context stack, or when the input context stack is empty.

You can use this parameter to specify what context asset binding is available for the component. This is useful if you want the component to switch between different input events.

For an example of working with contexts, see Input Request Bus (p. 650).

**Note**

Input event bus (EBus) messages aren't generated while the **Console** window is open.

---

**Working with the Input Component**

You can create an `.inputbindings` file for an **Input** component and specify the input values and events.

**Topics**

- Creating an Input to Event Binding Asset (p. 643)
- Creating Input Event Groups (p. 644)
- Creating Event Generators (p. 644)
- Mapping Input Events to Gameplay Actions (p. 645)
- Testing Your Input Events (p. 647)

---

**Creating an Input to Event Binding Asset**

Follow these steps to create an `.inputbindings` file.

**To create an input to event binding asset**

1. Create (p. 463) an entity.
2. In the **Perspective** viewport, select the entity.
3. In the **Entity Inspector**, click **Add Component**, and add the **Input** component.
4. In the **Entity Inspector** (p. 475), under **Input**, click the Input Bindings Editor icon to open the **Asset Editor**.
5. In the **Asset Editor**, choose **File, New, Input Bindings**.

6. Enter a name for the `.inputbindings` file and click **Save**.

### Creating Input Event Groups

After you create your `.inputbindings` file, you can add input event groups for different actions.

**To add an input event group**

1. In the **Asset Editor**, for your `.inputbindings` file, click the + icon.

2. For **Event Name**, enter a name for your event, such as **Action**.

### Creating Event Generators

After you create your input event group, you can add event generators to the group. An event generator is a handler that generates the named event. For example, a pressed key, a held mouse button, or a series of actions on a game controller results in the named event.

**To add an event generator to your input event group**

1. In the **Asset Editor**, for your input event group, next to **Event Generators**, click the + icon.

2. In the **Class to create** window, select **Input**, and click **OK**.

3. Specify your changes for the event generator. Each event generator has a set of properties that you can customize.

### Event Generator Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Device Type</td>
<td>The type of device that generates the input. Select from a list of available devices such as keyboard, mouse, and gamepad.</td>
</tr>
<tr>
<td>Input Name</td>
<td>A list of input options that depend on the selected input device type. For example, if you select <strong>keyboard</strong> for the <strong>Input Device Type</strong>, a list of possible keystrokes appear in this list, such as <code>keyboard_key_edit_space</code>.</td>
</tr>
<tr>
<td>Event value multiplier</td>
<td>A multiplier by which to scale the input value.</td>
</tr>
<tr>
<td>Dead zone</td>
<td>Specifies the threshold in which an event is generated.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
 | Values below this number are inactive and events are not generated.
 | Values above this number are active and events are generated.
 | For example, you can specify `gamepad` for the **Input Device Type** and `gamepad_thumbstick_r` for the **Input Name**.
 | If you specify a **Dead zone** value of 0.50, the thumbstick must move more than 50% to the right before the event is generated.

**Example**

The following `.inputbindings` file specifies a keyboard for the device type and the spacebar for the input name.

4. Save your `.inputbindings` file.
5. In the **Entity Inspector**, in the **Input** component, for **Input to event binding**, click the browse (…) icon and select your `.inputbindings` file.

**Mapping Input Events to Gameplay Actions**

After you create an `.inputbindings` file and specify your input events, you can use a Script Canvas graph or Lua script to map the input events to gameplay actions. You can create your graph in a visual scripting environment with the **Script Canvas** editor or write your own scripts with the Lua Editor (Lua IDE).

**Using a Script Canvas Graph for Input**

You can create a Script Canvas graph that maps to your input events. For more information, see Creating Gameplay with Script Canvas (p. 2420).

**To use a Script Canvas graph for input**

1. In the **Perspective** viewport, select the entity.
2. In the **Entity Inspector**, click **Add Component** and add the **Script Canvas** component.
3. In the **Script Canvas** component, for **Script Canvas Asset**, specify a Script Canvas graph like the following.
Example Script Canvas Graph

In the following graph, the Input Handler node maps the Event Name Action to the .inputbindings file.

Using a Lua Script for Input

You can also create a Lua script that maps to your input events. For more information about Lua, see Writing Lua Scripts (p. 2683).

To add a Lua script for input

1. In the Perspective viewport, select the entity.
2. In the Entity Inspector, click Add Component, and add the Lua Script (p. 683) component.
3. In the Lua Script component, specify a Lua script file like the following.
Example Lua Script

The following Lua script maps the Event Name Action to the .inputbindings file.

```lua
local tutorial_input = {
  Properties = {
  },
}

function tutorial_input:OnActivate()
  self.Inputs = {
    self.Inputs.OnPressed = function(_, value)
      TransformBus.Event.SetLocalScaleZ(self.entityId, 2.0)
    end
    self.Inputs.OnHeld = function (_, value)
      TransformBus.Event.RotateAroundLocalZ(self.entityId, 0.01)
    end
    self.Inputs.OnReleased = function (_, value)
      TransformBus.Event.SetLocalScaleZ(self.entityId, 1.0)
    end
    self.InputNotificationBus = InputEventNotificationBus.Connect( inputNotificationId("Action"))
  end

  function tutorial_input:OnDeactivate()
    self.InputNotificationBus:Disconnect()
  end

  return tutorial_input
```

Testing Your Input Events

After you specify the Script Canvas graph or Lua script, you can test your input events.

To test your input events

1. In the Perspective viewport, select your entity.
2. In the Entity Inspector, click Add Component, and add the Mesh (p. 684) component.
3. For Mesh asset, specify a mesh asset file. This gives your entity a shape. For example, you can specify the lumberyard_version\dev\SamplesProject\Objects\Primitives\cube_001.cgf file.
4. Press Ctrl+G to enter game mode.
5. Press the keyboard spacebar so that your entity rotates on the local z-axis.
Example

6. To exit gameplay mode, press Esc

Input Component EBus Interface

Input subcomponents are objects that have the same lifetime as components and must override Activate and Deactivate.

```cpp
// InputSubComponent
void Activate(const AZ::InputEventNotificationId& channelId) override;
void Deactivate(const AZ::InputEventNotificationId& channelId) override;
```

You can use the GameplayNotificationBus to work with the InputSubComponent. For more information, see GameplayNotifications (p. 1031).
You can find example Lua scripts and code in the `lumberyard_version\dev\Gems\StartingPointInput\Assets\Scripts\Input` directory.

**Input Event Notification Bus**

Use the following notification functions with the event notification bus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnPressed</td>
<td>Event sent when an input surpasses the threshold.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>OnHeld</td>
<td>Event sent when an input continues to surpass the threshold.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>OnReleased</td>
<td>Event sent when an input no longer surpasses the threshold.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

```lua
local held = 
{
    Properties = 
    {
        IncomingInputEventName = "",
        OutgoingGameplayEventName = "",
    },
}

function held:OnActivate()
    local inputBusId = InputEventNotificationId(self.Properties.IncomingInputEventName)
    self.inputBus = InputEventNotificationBus.Connect(self, inputBusId)
end

function held:OnHeld(floatValue)
end

function held:OnReleased(floatValue)
    GameplayNotificationBus.Event.OnEventEnd(GameplayNotificationId(self.entityId, self.Properties.OutgoingGameplayEventName), floatValue)
end

function held:OnDeactivate()
    self.inputBus:Disconnect()
end

return held
```
**Input Request Bus**

Use the following functions with the input request bus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>PushContext</td>
<td>Pushes a new context onto the stack, which becomes the active context.</td>
<td>String</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>PopContext</td>
<td>Removes the top context from the input context stack.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>PopAllContexts</td>
<td>Clears the context stack, and the active context becomes the empty string: &quot;&quot;</td>
<td>None</td>
<td>Empty string</td>
<td>Yes</td>
</tr>
<tr>
<td>GetCurrentContext</td>
<td>Returns the context at the top of the stack. If the stack is empty, returns: &quot;&quot;</td>
<td>None</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example Input Contexts**

You can use the **Input contexts** parameter to specify what context asset binding is available. For example, you can switch contexts so that when a character is under water, the input events are different than when the character is on the ground.

For example, for the **Input** component, you can specify **Input contexts**, such as an empty string, "under water", and "run". When you add a **Lua** component to the entity, you can specify the different **Input contexts**, "under water", and "run" in your Lua script.

This means that when your Lua script runs, the Lua script tells the **Input** component which context to use.

```lua
local foo = {
  Properties = {
    Context = {
      default = "", description = "A context to push onto the input stack. Like console, or UI, or turret, under water, etc."},
    }
}{}

function foo:OnActivate()
  -- by default the context is blank ""
  InputRequestBus.Broadcast.PushContext(self.Properties.Context) -- context stack is now 1)user defined property
```
Legacy Terrain level component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Legacy Terrain level component enables the legacy terrain system. This component works as a switch. When the Legacy Terrain level component is added to a level, terrain is displayed and can be edited. When the Legacy Terrain level component is removed from the level, terrain is not displayed and cannot be edited. Adding or removing the Legacy Terrain level component is a non-destructive operation. Existing terrain heightmap and texture data remains intact when the Legacy Terrain level component is removed from the level.

Note
The Legacy Terrain component is a level component and is added to the level through the Level Inspector.

To enable the Legacy Terrain system in a level

1. In the Lumberyard Editor, open the Level Inspector by choosing its tab, or by choosing Level Inspector from Tools in the main menu bar.
2. In the **Level Inspector**, choose **Add Component**. In the components list, choose **Legacy Terrain**.

The terrain is displayed in the **Perspective** view.

**Note**
The **Legacy Terrain** level component has no properties because it is a switch that enables the legacy terrain system.

### Resize legacy terrain

Terrain size is defined by a *heightmap*, which is a gray scale texture that contains elevation information, and a *meters per texel* property. The meters per texel property sets the relationship between texels in the heightmap and the terrain size in world space. A **1024 x 1024** heightmap with 1 meter per texel creates a terrain that's 1.024 square kilometers in size. Each texel in the heightmap defines one square meter of the terrain.

#### To resize terrain

1. In the Lumberyard Editor main menu bar, open the **Terrain Editor** by choosing **Terrain Editor** from **Tools** in the main menu bar.
2. In the main menu bar for the **Terrain Editor**, choose **Resize Terrain** from the **Modify** menu.
3. Choose from the lists to set the **Heightmap Resolution** and **Meters Per Texel** properties for the legacy terrain.

### Generate a legacy terrain texture

Terrain textures for the legacy terrain system are generated within Lumberyard Editor.
To generate a legacy terrain texture

1. In the Lumberyard Editor, main menu bar, choose the Game menu.
2. In the Game menu, choose the Terrain group, and then Generate Terrain Texture.
3. Select one of the Texture Dimensions options in the Generate Terrain Texture window and choose OK.

Use PhysX or legacy physics with legacy terrain

If your project uses legacy physics components, interactions between the terrain and physics entities will work without additional steps.

If your project uses the PhysX system, you must add a PhysX Terrain level component to enable PhysX entities to interact with the terrain. The PhysX Terrain level component is added in the Level Inspector with the Legacy Terrain level component.
To enable PhysX with the Legacy Terrain system in a level

1. In the Lumberyard Editor, open the Level Inspector by choosing its tab, or by choosing Level Inspector from Tools in the main menu bar.
2. Choose Add Component in the Level Inspector and select PhysX Terrain from the components list.

Important
For historical reasons, the PhysX Terrain component can be added to the level via the Entity Inspector but this is not recommended. This functionality has been maintained for compatibility. The PhysX Terrain component should not be instantiated more than once in a level. In the future and for new projects, use the Level Inspector to add the PhysX Terrain level component to the level.

Lens Flare

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The **Lens Flare** component allows the placement of a lens flare on an entity.

## Lens Flare Component Properties

The **Lens Flare** component has the following properties:

**Visible**
- If selected, shows the lens flare.

**Library**
- Select a lens flare library that has been authored by the **Lens Flare** editor.

**Lens flare**
- Select a lens flare from the available flares in the lens flare library.

**Flare Settings**

**Minimum spec**
- The minimum spec at which this lens flare is enabled.
  - Default: Low

**FOV**
- The field of view (FOV) in degrees around the lens flare. Use **360** degrees to make the lens flare visible from all angles.
  - Default: 360

**Size**
- The size of the lens flare.

**Attach to sun**
- If selected, attaches the lens flare to the sun (as opposed to attaching to the entity).

**Ignore vis areas**
- If selected, lens flare ignores vis areas.

**Indoor only**
- If selected, lens flare is rendered indoors only.

**On initially**
- If selected, the lens flare is on when the scene starts.

**View distance multiplier**
- Adjust the maximum view distance. For example, **1.0** would use the default and **1.1** would be 10% further than the default.

**Color Settings**

**Tint**
- Color of the lens flare.
Lens Flare

Tint [alpha]

Alpha value that sets the flare's transparency.

Brightness

Brightness of the lens flare.

Animation

Sync with light

If selected, uses the animation settings of the provided light. Select the light entity in the **Light** setting.

Light

This setting appears when the **Sync with light** setting is selected. Use the picker (hand icon) to select the light component you want to sync animation settings with.

Style

Light animation curve ID (style) as it corresponds to values in `Light.cfx`.

Speed

Multiple of the base animation rate.

Phase

Animation start offset from 0 to 1. 0.1 would be 10% into the animation.

**EBus Request Bus Interface**

Use the following request function with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

- `SetLensFlareState` (on or off) – Turns the lens flare on or off.
- `TurnOnLensFlare` – Turns the lens flare on.
- `TurnOffLensFlare` – Turns the lens flare off.
- `ToggleLensFlare` – Toggles the lens flare state (on to off, or off to on).

**EBus Notification Bus Interface**

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

- `LensFlareTurnedOn` – Sends a signal when the lens flare is turned on.
- `LensFlareTurnedOff` – Sends a signal when the lens flare is turned off.

The following is an example of script using the **Request Bus Interface**.
function example:OnActivate()
    LensFlareComponentRequestBus.Event.SetLensFlareState(self.entityId, LensFlareComponentState.Off)
    LensFlareComponentRequestBus.Event.TurnOnLensFlare(self.entityId)
    LensFlareComponentRequestBus.Event.TurnOffLensFlare(self.entityId)
    LensFlareComponentRequestBus.Event.ToggleLensFlare(self.entityId)
end

### Lightning

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Lightning** component to create a single lightning bolt effect. In Lumberyard Editor, you can add the **Lightning** component to entities and trigger them later from Lua or Script Canvas. For best results, use the **Lightning** component as part of a dynamic slice along with a spawning system. After the **Lightning** component finishes its effect, the entity is automatically destroyed. This makes it easier for you to spawn entities with a **Lightning** component, without needing manually to manage the lifetime of each entity.

**Note**

To enable the **Lightning** component, you must enable the Lightning gem. For more information, see Lightning Arc Gem (p. 1152).

The **Lightning** component does not produce visual effects on its own. The component schedules timings with a specified set of entities. You can use the following components to create the lightning bolt effect:

- **Particle** (p. 702) component emitter to create the lightning bolt.
- A **Light** component for the dynamic lighting and shadowing. You can use any of the light components, but the **Point Light** (p. 706) component is recommended.
- **Sky Highlight** (p. 847) component for the bright flash in the sky.
- **Audio Trigger** (p. 559) and **Audio Proxy** (p. 557) components for the rumbling of thunder.
- The **Lightning** component to manage the lifetimes and intensities of the other entities to create a flash of lightning.

**Example**

The **Lightning** component randomly spawns in a small area.
Contents

- Lightning Component Properties (p. 659)
  - Lightning Bolt (p. 659)
  - SkyHighlight (p. 659)
  - Light (p. 659)
  - Audio (p. 660)
- EBus Request Bus Interface (p. 660)
  - StartEffect (p. 660)
  - SetStartOnActivate (p. 660)
  - GetStartOnActivate (p. 661)
  - SetRelativeToPlayer (p. 661)
  - GetRelativeToPlayer (p. 661)
  - SetLightningParticleEntity (p. 661)
  - GetLightningParticleEntity (p. 662)
  - SetLightEntity (p. 662)
  - GetLightEntity (p. 662)
  - SetSkyHighlightEntity (p. 662)
  - GetSkyHighlightEntity (p. 663)
  - SetAudioEntity (p. 663)
  - GetAudioEntity (p. 663)
  - SetSpeedOfSoundScale (p. 664)
  - GetSpeedOfSoundScale (p. 664)
  - SetLightRadiusVariation (p. 664)
  - GetLightRadiusVariation (p. 665)
  - SetLightIntensityVariation (p. 665)
• GetLightIntensityVariation (p. 665)
• SetParticleSizeVariation (p. 665)
• GetParticleSizeVariation (p. 666)
• SetLightningDuration (p. 666)
• GetLightningDuration (p. 666)
• Request Bus Example Script (p. 667)

Lightning Component Properties

The Lightning component has the following properties:

Start on Activate

Specifies whether lightning effects start when the Light component activates.

Relative to Player

Specifies whether the lightning effects are relative to the current active camera.

Duration

Amount of time in seconds that the lightning strike lasts.

The particle effect lasts for the specified duration; the lighting effects can flash multiple times depending on how quickly the light intensity disappears.

Lightning Bolt

The Lightning component has the following lightning bolt options:

Particle Entity

(Optional) An entity with a Particle (p. 702) component that creates a lightning bolt effect. This particle can be emitted multiple times depending on the duration of the effect.

Size Variation

Specifies how much of the particle entity's global size parameter is randomly modified. For more information, see SetParticleSizeVariation (p. 665).

SkyHighlight

The Lightning component has following sky highlight options:

Sky Highlight Entity

(Optional) An entity with a Sky Highlight component that creates a flash of light in the sky. The properties of the Sky Highlight component are respected.

Light

The Lightning component has following light options:

Light Entity

(Optional) An entity with a Light component that creates dynamic lightning. The properties of the Light component are the same, but its intensity and radius are modified.
(Point Light (p. 706) is recommended)

**Radius Variation**

Specifies how much the light's radius is randomly modified. For more information, see `SetLightRadiusVariation` (p. 664).

**Intensity Variation**

Specifies how much the light's intensity is randomly modified. For more information, see `SetLightIntensityVariation` (p. 665).

**Audio**

The **Lightning** component has following audio options:

**Audio Entity**

An entity with the **Audio Trigger** (p. 559) and **Audio Proxy** (p. 557) components attached. These components are required if you want to enable audio. You can also add an optional **Audio Rtpc** (p. 557) component. The audio components plays the audio, depending on the distance away from the listener.

**Speed of Sound Scale**

Specifies how long it takes for audio to arrive at the listener, in seconds. For more information, see `SetSpeedOfSoundScale` (p. 664).

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game. You can use this EBus to communicate to an entity with a **Lightning** component attached. The EBus is available at game run time and editing and can be accessed from C++, Lua, and the **Script Canvas** editor.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**StartEffect**

Enables the lightning effect.

**Parameters**

None

**Return**

None

**Scriptable**

Yes

**SetStartOnActivate**

Specify if the lightning effect starts when the **Lightning** component activates.

**Note**

This function is useful only during edit time.
Parameters
   Type: Boolean
Return
   None
Scriptable
   Yes

GetStartOnActivate
Returns true if the lightning effect starts when the Lightning component activates.
Parameters
   None
Return
   Type: Boolean
Scriptable
   Yes

SetRelativeToPlayer
Specify if the start of the lightning effect is relative to the player camera.
Parameters
   Type: Boolean
Return
   None
Scriptable
   Yes

GetRelativeToPlayer
Returns true if the start of the lightning effect is relative to the player camera.
Parameters
   Type: Boolean
Return
   None
Scriptable
   Yes

SetLightningParticleEntity
Sets the entity that contains the Particle (p. 702) component that is used for the lightning particle. The entity is destroyed when the effect finishes.
Lumberyard User Guide

Lightning

Parameters
Type: AZ::EntityId

Return
None

Scriptable
Yes

GetLightningParticleEntity
Retuns the entity that contains the Particle (p. 702) component.

Parameters
None

Return
Type: AZ::EntityId

Scriptable
Yes

SetLightEntity
Sets the entity that contains the Light component, which is used for lightning effects.
The entity is destroyed when the effect finishes.

Parameters
Type: AZ::EntityId

Return
None

Scriptable
Yes

GetLightEntity
Returns the entity that contains the Light component.

Parameters
None

Return
Type: AZ::EntityId

Scriptable
Yes

SetSkyHighlightEntity
Sets the entity that has the Sky Highlight component for this effect.
The entity is destroyed when the effect finishes.

**Parameters**

None

**Return**

Type: AZ::EntityId

**Scriptable**

Yes

**GetSkyHighlightEntity**

Returns the entity that has the **Sky Highlight** component.

**Parameters**

None

**Return**

Type: AZ::EntityId

**Scriptable**

Yes

**SetAudioEntity**

Sets the entity that has the audio component for this effect.

To enable audio, the entity must have the **Audio Trigger (p. 559)** and **Audio Proxy (p. 557)** components and an optional **Audio Rtpc (p. 557)** component. If an audio component is missing, the lightning effect may not work as expected.

The entity is destroyed when the effect finishes.

**Parameters**

Type: AZ::EntityId

**Return**

None

**Scriptable**

Yes

**GetAudioEntity**

Returns the entity that has the audio component.

**Parameters**

None

**Return**

Type: AZ::EntityId
Scriptable

Yes

SetSpeedOfSoundScale

Sets the speed of sound scale for this effect. You can use this parameter if want the audio to take more or less time to reach the player and you don't want to move your lightning effect.

Note
Sound travels at 340.29 meters per second. A value of 0.5 makes sound from this effect take half as much time to reach the player.

Parameters

Type: Float

Return

None

Scriptable

Yes

GetSpeedOfSoundScale

Returns the speed of sound scale for this effect.

Parameters

None

Return

Type: Float

Scriptable

Yes

SetLightRadiusVariation

Sets the amount of random variation to apply to the light's radius.

This variation is a percentage of the light's radius. A value of 0.2 means that between ~20% and 20% of the light's radius value is added back to the light's radius.

For example, the light radius is 5.0, and variation is 0.2; this means that the range of the light radius is: 5.0 +/− (5.0 * 0.2) or 5.0 +/− 20% of 5.0.

Parameters

Type: Float

Return

None

Scriptable

Yes
**GetLightRadiusVariation**

Returns the amount of random variation to apply to the light's radius.

**Parameters**

None

**Return**

Type: Float

**Scriptable**

Yes

**SetLightIntensityVariation**

Sets the amount of random variation to apply the Light component's diffuse and specular multipliers.

The Light component calculates the light intensity based on the duration of the lightning effect. This value is a percentage of that light intensity value. A value of 0.2 means that between −20% and 20% of the light intensity value is added back to the light's diffuse and specular multipliers.

For example, the light intensity is 10.0, and variation is 0.3; this means that the range of the light radius is: 10.0 +/− (10.0 * 0.3) or 10.0 +/− 30% of 10.0.

**Parameters**

Type: Float

**Return**

None

**Scriptable**

Yes

**GetLightIntensityVariation**

Returns the amount of random variation to apply the light's diffuse and specular multipliers.

**Parameters**

None

**Return**

Type: Float

**Scriptable**

Yes

**SetParticleSizeVariation**

Sets the amount of random variation to apply to the particle's size.

This variation is a percentage of the Particle (p. 702) component's Global size scale parameter. A value of 0.2 means that between −20% and 20% of the particle's global size value is added back to the size of the emitted particles.
For example, the particle size is $1.0$, and variation is $0.5$; the range of the light radius is: $1.0 +/-(1.0 * 0.5)$ or $1.0 +/-$ $50\%$ of $1.0$.

**Parameters**

- Type: Float

**Return**

- None

**Scriptable**

- Yes

---

**GetParticleSizeVariation**

Returns the amount of random variation to apply to the particle's size.

**Parameters**

- None

**Return**

- Type: Float

**Scriptable**

- Yes

---

**SetLightningDuration**

Sets how long the lightning strike lasts, in seconds. This value determines how long the lightning particle effect lasts. During this time, the sky highlight and light can continue to flash.

**Parameters**

- Type: Double

**Return**

- None

**Scriptable**

- Yes

---

**GetLightningDuration**

Returns how long the lightning strike lasts, in seconds.

**Parameters**

- None

**Return**

- Type: Double

**Scriptable**

- Yes
Request Bus Example Script

```lua
function example:OnActivate()
    LightningComponentRequestBus.Event.StartEffect(self.entityId)
end
```

Lighting Arc

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Lightning Arc component to create short arcs (also called sparks) between an emitter and a series of target entities. When an arc is emitted, the arc jumps between the center of the entity to which the Lightning Arc component is attached and the center of the target entity.

To enable the Lightning Arc component, you must enable the Lightning Arc gem. For more information, see Lightning Arc Gem (p. 1152).

Example

The Lightning Arc component is attached to the entity in the center, and the arc randomly jumps between three targets. This example uses the default lightning material in the Lightning Arc gem and the Tesla_core arc preset. For more information, see Customizing a Lightning Arc Preset (p. 1156).
Lightning Arc Component Properties

The Lightning Arc component has the following properties:

**Enabled**

Specifies whether the Lightning Arc component emits arcs.

**Targets**

The collection of targets to which the Lightning Arc component emits arcs.

**Material**

The lightning arc material. This parameter does not require a specific type of material, but it's recommended that you use the Illum shader (p. 1697) and a texture with opacity.

You can find the default files in the following directory: Gems\LightningArc\Assets\materials\effects

Default value: None

**Arc Preset Name**

The name of the arc preset in use; the arc preset is specified in the lightningarceffects.xml file. If you change any of the arc parameters in this file, this field changes to <Custom>.

You can find the available arc preset names in the following directory: Gems\LightningArc\Assets\libs\lightningarc\lightningarceffects.xml

**Refresh Presets**

If you make changes to the lightningarceffects.xml file, click Refresh Presets to get the most recent arc preset names.

**Arc Parameters**

The Lightning Arc component has the following lightning options:

**Contents**

- Lightning (p. 668)
- Branch (p. 669)
- Strike (p. 669)
- Beam (p. 669)

**Lightning**

**Deviation**

Amount of deviation applied to the arcs. Specify lower values to make the arc appear more smooth.

**Fuzziness**

Amount of noise applied to the arcs.
Velocity

Specify how fast an arc drifts upward after it is emitted.

Branch

The **Lightning Arc** component has following branch options:

Max Level

Maximum number of branches that can spawn from an arc. For more information, see `SetBranchMaxLevel (p. 679)`.

Probability

Specifies how likely that a child branch spawns from an arc. For more information, see `SetBranchProbability (p. 678)`.

Strike

The **Lightning Arc** component has following strike options:

Time Min

Minimum amount of time that an arc is kept alive.

Time Max

Maximum amount of time that an arc is kept alive.

Fade Out

Specifies how long it takes for an arc to fade out.

Segment Count

Number of segments in an arc. Specify more segments to make the arcs appear more winding.

Point Count

Number of points for each segment. Specify more points to increase the noise effect that is defined by the **Fuzziness** parameter.

Max Strike Count

Specify how many arcs can be alive at one time from this **Lightning Arc** component.

Beam

The **Lightning Arc** component has following beam options:

Size

The width of the arcs that the **Lightning Arc** component creates. The branches that spawn off the arc beam will have half this size.

Tex Tiling

Texture tiling based on the world size of the arc beam. For more information, see `SetBeamTexTiling (p. 680)`.

Tex Shift

Specify how fast to move through textures in the arc's animation. For more information, see `SetBeamTexShift (p. 681)`.
**Tex Frames**

Specify how many frames are in the arc's animation.

**Tex FPS**

Specify how many frames per second are in the arc's animation.

### Timing

The **Lightning Arc** component has following timing options:

**Delay**

Time between emitted arcs, in seconds.

**Delay Variation**

Variation in time between emitted arcs. For more information, see [SetDelayVariation](#).

### EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game. You can use this EBus to communicate to an entity with a **Lightning Arc** component attached. The EBus is available at game run time and editing and can be accessed from C++, Lua, and the **Script Canvas** editor.

For more information about using the event bus (EBus) interface, see [Working with the Event Bus (EBus) system](#).

**Contents**

- *Enable* (p. 671)
- *Disable* (p. 671)
- *Toggle* (p. 672)
- *IsEnabled* (p. 672)
- *SetTargets* (p. 672)
- *GetTargets* (p. 672)
- *SetDelay* (p. 673)
- *GetDelay* (p. 673)
- *SetDelayVariation* (p. 673)
- *GetDelayVariation* (p. 674)
- *SetStrikeTimeMin* (p. 674)
- *GetStrikeTimeMin* (p. 674)
- *SetStrikeTimeMax* (p. 674)
- *GetStrikeTimeMax* (p. 675)
- *SetStrikeFadeOut* (p. 675)
- *GetStrikeFadeOut* (p. 675)
- *SetStrikeSegmentCount* (p. 675)
- *GetStrikeSegmentCount* (p. 676)
- *SetStrikePointCount* (p. 676)
- *GetStrikePointCount* (p. 676)
• **SetLightningDeviation** (p. 677)
• **GetLightningDeviation** (p. 677)
• **SetLightningFuzziness** (p. 677)
• **GetLightningFuzziness** (p. 677)
• **SetLightningVelocity** (p. 678)
• **GetLightningVelocity** (p. 678)
• **SetBranchProbability** (p. 678)
• **GetBranchProbability** (p. 679)
• **SetBranchMaxLevel** (p. 679)
• **GetBranchMaxLevel** (p. 679)
• **SetMaxStrikeCount** (p. 679)
• **GetMaxStrikeCount** (p. 680)
• **SetBeamSize** (p. 680)
• **GetBeamSize** (p. 680)
• **SetBeamTexTiling** (p. 680)
• **GetBeamTexTiling** (p. 681)
• **SetBeamTexShift** (p. 681)
• **GetBeamTexShift** (p. 681)
• **SetBeamTexFrames** (p. 682)
• **GetBeamTexFrames** (p. 682)
• **SetBeamTexFPS** (p. 682)
• **GetBeamTexFPS** (p. 682)
• **Request Bus Example Script** (p. 683)

### Enable

Enables the component to emit lightning arcs.

**Parameters**

None

**Return**

None

**Scriptable**

Yes

### Disable

Disables lightning arc emission.

**Parameters**

None

**Return**

None
**Scriptable**

Yes

**Toggle**

Toggles lightning arc emission.

**Parameters**

None

**Return**

None

**Scriptable**

Yes

**IsEnabled**

Returns whether the **Lightning Arc** component emits lightning arcs.

**Parameters**

None

**Return**

Type: Boolean

**Scriptable**

Yes

**SetTargets**

Sets the target entities to which the **Lightning Arc** component emits arcs.

**Parameters**

AZStd::vector<AZ::EntityId>

Type: Vector of entity IDs.

**Return**

None

**Scriptable**

Yes

**GetTargets**

Returns the target entities to which the **Lightning Arc** component emits arcs.

**Parameters**

None
Return

`AZStd::vector<AZ::EntityId>`

Type: Vector of entity IDs.

Scriptable

Yes

SetDelay

Sets the time between emitted arcs, in seconds.

Parameters

Type: Double

Return

None

Scriptable

Yes

GetDelay

Returns the time between emitted arcs, in seconds.

Parameters

None

Return

Type: Double

Scriptable

Yes

SetDelayVariation

Sets the variation in time between emitted arcs.

Delay variation is a random range applied to the `Delay` parameter to calculate the time that the next arc emits.

The random variation is in the following range: `[delay variation * 0.5, delay variation]`.

For example, if the delay is `2.0` and the delay variation is `1.0`, the range of time between arc emissions is `[2.5 and 3.0]`.

Parameters

Type: Double

Return

None
Scriptable
Yes

**GetDelayVariation**

Returns the variation in time between emitted arcs.

**Parameters**
None

**Return**
Type: Double

Scriptable
Yes

**SetStrikeTimeMin**

Sets the minimum amount of time that an arc is kept alive.

**Parameters**
Type: Float

**Return**
None

Scriptable
Yes

**GetStrikeTimeMin**

Returns the minimum amount of time that an arc is kept alive.

**Parameters**
None

**Return**
Type: Float

Scriptable
Yes

**SetStrikeTimeMax**

Sets the maximum amount of time that an arc is kept alive.

**Parameters**
Type: Float

**Return**
None
Scriptable
Yes

GetStrikeTimeMax
Returns the maximum amount of time that an arc is kept alive.

Parameters
None

Return
Type: Float

Scriptable
Yes

SetStrikeFadeOut
Sets how long it takes for an arc to fade out.

Note
Lumberyard completes this fade out internally by scaling the size to 0 rather than adjusting the transparency.

Parameters
Type: Float

Return
None

Scriptable
Yes

GetStrikeFadeOut
Returns how long it takes for an arc to fade out.

Parameters
None

Return
Type: Float

Scriptable
Yes

SetStrikeSegmentCount
Sets the number of segments in an arc.

Specify more segments to make the arcs appear more winding. A Lightning arc component must have a minimum of 1 segment. If you specify 0, a warning appears and 1 is used instead.
Parameters
  Type: AZ::u32

Return
  None

Scriptable
  Yes

GetStrikeSegmentCount
Returns the number of segments in an arc.

Parameters
  None

Return
  Type: AZ::u32

Scriptable
  Yes

SetStrikePointCount
Sets the number of points for each segment.

Note
Specify more points to increase the noise effect that is defined by the Fuzziness parameter. A Lightning arc component must have a minimum of 1 point. If you specify 0, a warning appears and 1 is used instead.

Parameters
  Type: AZ::u32

Return
  None

Scriptable
  Yes

GetStrikePointCount
Returns the number of points for each segment in an arc.

Parameters
  None

Return
  Type: AZ::u32

Scriptable
  Yes
SetLightningDeviation

Sets the amount of deviation applied to the arcs.

**Note**
Specify lower values to make the arc appear more smooth.

**Parameters**
Type: Float

**Return**
None

**Scriptable**
Yes

GetLightningDeviation

Returns the amount of deviation applied to the arcs.

**Parameters**
None

**Return**
Type: Float

**Scriptable**
Yes

SetLightningFuzziness

Sets the amount of noise applied to the arcs.

**Parameters**
Type: Float

**Return**
None

**Scriptable**
Yes

GetLightningFuzziness

Returns the amount of noise applied to the arcs.

**Parameters**
None

**Return**
Type: Float
Scriptable

Yes

SetLightningVelocity
Sets how fast an arc drifts upwards after it emits.

Parameters
Type: Float

Return
None

Scriptable

Yes

GetLightningVelocity
Returns how fast an arc drifts upwards after it emits.

Parameters
None

Return
Type: Float

Scriptable

Yes

SetBranchProbability
Sets how likely a branch spawns off of an arc.

A branch (or child arc) is an arc that is half the size and intensity of the parent arc. A branch has the same emission point and target as its parent.

- A value of 0 means that branches do not spawn.
- A value of 0.5 means there is a 50% chance of spawning a branch.
- A value of 2.0 means there is a 100% chance of spawning two branches.

For more information about setting the maximum number of branches, see SetBranchMaxLevel (p. 679).

Parameters
Type: Float

Return
None

Scriptable

Yes
**GetBranchProbability**

Returns how likely a branch spawns off the parent arc.

**Parameters**

None

**Return**

Type: Float

**Scriptable**

Yes

**SetBranchMaxLevel**

Sets the maximum number of branches that can spawn off the parent arc.

- A value of 0 means branches do not spawn, regardless of branch probability.
- A value of 3 means that between 0 and 3 branches can spawn, depending on the branch probability.

**Parameters**

Type: AZ::u32

**Return**

None

**Scriptable**

Yes

**GetBranchMaxLevel**

Returns the maximum number of branches allowed to spawn off the parent arc.

**Parameters**

None

**Return**

Type: AZ::u32

**Scriptable**

Yes

**SetMaxStrikeCount**

Sets how many arcs can be alive at one time from this Lightning Arc component; this includes parent and branch arcs.

**Parameters**

Type: AZ::u32
Return

None
Scriptable
Yes

GetMaxStrikeCount

Returns the maximum number of arcs that can be alive at one time from this *Lightning Arc* component.

Parameters

None
Return

Type: AZ::u32
Scriptable
Yes

SetBeamSize

Sets the width of the generated arcs. Branch arcs be half this size.

Parameters

Type: Float
Return

None
Scriptable
Yes

GetBeamSize

Returns the width of the generated arcs.

Parameters

None
Return

Float
Scriptable
Yes

SetBeamTexTiling

Sets the texture tiling based on the world size of the arc beam.

- A value of 2.0 means that the texture wraps around twice for each meter.
- A value of 0.25 means that the texture wraps around four times for each meter.
Note
Only the U coordinate of the texture map is affected by this parameter.

Parameters
Type: Float

Return
None

Scriptable
Yes

GetBeamTexTiling
Returns the texture tiling parameter for the arc beam.

Parameters
None

Return
Type: Float

Scriptable
Yes

SetBeamTexShift
Sets how fast to move through textures in the arc's animation.

Note
The U value of the texture coordinate moves at this specified rate. The V value is automatically calculated to select the correct frame.

Parameters
Type: Float

Return
None

Scriptable
Yes

GetBeamTexShift
Returns how fast to move through textures in the arc's animation.

Parameters
None

Return
Type: Float
SetBeamTexFrames
Sets how many frames are in an arc's animation.

Parameters
Type: Float

Return
None

GetBeamTexFrames
Returns the number of frames in an arc's animation.

Parameters
None

Return
Type: Float

SetBeamTexFPS
Sets how many frames per second are in an arc's animation.

Parameters
Type: Float

Return
None

GetBeamTexFPS
Returns how many frames per second are in an arc's animation.

Parameters
None

Return
Type: Float
Scriptable

Yes

Request Bus Example Script

```lua
function example:OnActivate()

-- Send some events to a LightningArcComponent attached to the same entity as this script
  LightningArcComponentRequestBus.Event.Toggle(self.entityId)
  LightningArcComponentRequestBus.Event.SetDelayVariation(self.entityId, 0.4)
end
```

EBus Notification Bus Interface

Use the following notification functions with the Lightning Arc component notification EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

OnSpark

Notifies an event that triggers when the specified component fires a spark.

Parameters

None

Return

None

Scriptable

Yes

Notification Bus Example Script

```lua
function example:OnActivate()

  -- Listen for lightning arc notifications on the same entity to which this script is attached
  self.lightningArcHandler = LightningArcComponentNotificationBus.Connect(self, self.entityId)
end

function example:OnSpark()

  Debug.Log("On Spark Triggered")
end

function example:OnDeactivate()

  self.lightningArcHandler:Disconnect()
end
```

Lua Script

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use the **Lua Script** component to add arbitrary Lua logic to an entity in the form of a Lua script. For information on the **Lua Script** component and writing Lua scripts, see **Writing Lua Scripts (p. 2683)**.

## Mesh

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can add the **Mesh** component to an entity to provide visual geometry on an entity. The **Mesh** component has key controls and options for Lumberyard's basic rendering features. The supported geometry type is meshes (`.cgf`).

### Mesh Component Properties

![Mesh component properties](image)

The **Mesh** component has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Entity is visible.</td>
</tr>
<tr>
<td>Mesh asset</td>
<td>Specifies the asset file for the mesh entity.</td>
</tr>
<tr>
<td>Material override</td>
<td>Specifies an override material.</td>
</tr>
<tr>
<td>Opacity</td>
<td>The entity's degree of transparency.</td>
</tr>
<tr>
<td>Max view distance</td>
<td>Maximum distance from which this entity can be viewed.</td>
</tr>
<tr>
<td>View distance multiplier</td>
<td>Adjusts the maximum view distance. If set to 1.0, then the default maximum view distance is used. For example, 1.1 extends the default by 10%.</td>
</tr>
<tr>
<td>LOD distance ratio</td>
<td>Sets the level of detail (LOD) ratio over distance.</td>
</tr>
<tr>
<td>Cast shadows</td>
<td>Casts shadow maps.</td>
</tr>
<tr>
<td>Use VisAreas</td>
<td>Allows vis areas to control the component's visibility.</td>
</tr>
</tbody>
</table>
Advanced

The **Mesh** component has different advanced properties that depend on whether your mesh is static or dynamic.

- A mesh is dynamic when the **Static** property is not set on the **Transform** component. This is the default setting when you create an entity. Dynamic meshes do not affect nav meshes.
- A mesh is static when the **Static** property is set on the **Transform** component. You can use static objects for more optimal paths during rendering and processing. We recommend that you create static meshes whenever possible. A static mesh can move or deform only when the **Receive Wind** or **Deformable Mesh** properties are set.

For more information, see [Transform Component Properties (p. 879)](#).

### Advanced Properties for Dynamic Meshes

If your mesh is dynamic, the **Mesh** component has the following advanced properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive wind</td>
<td>Affected by wind.</td>
</tr>
<tr>
<td>Accept decals</td>
<td>Receives decals.</td>
</tr>
<tr>
<td>Deformable mesh</td>
<td>Allows static mesh assets to deform that have specific dynamic data. For example, you can have a mesh deform asset that is stationary.</td>
</tr>
</tbody>
</table>

### Advanced Properties for Static Meshes

If the mesh is static, the **Mesh** component has the following advanced properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rain occluder</td>
<td>Blocks or stops dynamic raindrops.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Affect dynamic water</td>
<td>Generates ripples in dynamic water.</td>
</tr>
<tr>
<td>Receive wind</td>
<td>Affected by wind.</td>
</tr>
<tr>
<td>Accept decals</td>
<td>Receives decals.</td>
</tr>
<tr>
<td>Affect navmesh</td>
<td>Affects navmesh generation.</td>
</tr>
<tr>
<td>Visibility occluder</td>
<td>Blocks visibility of other objects.</td>
</tr>
<tr>
<td>Deformable mesh</td>
<td>Allows static mesh assets to deform that have specific dynamic data. For example, you can have a mesh deform asset that is stationary.</td>
</tr>
<tr>
<td>Affects GI</td>
<td>Affects global illumination results.</td>
</tr>
</tbody>
</table>

**Note**

If you select the Receive wind or Deformable mesh settings, the entity's Transform component remains Static, but the mesh is considered dynamic.

**Navigation**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Navigation component provides path-finding and path-following functionality for AI movement, typically on a navigation mesh.

**Topics**
- Navigation Component Properties (p. 687)
Navigation Component Properties

The Navigation component has the following properties:

Agent Type

Specifies this AI's entity type for navigation purposes. Defining the agent type determines which navigation area (p. 693) the entity follows in a scenario where there are different navigation meshes for larger vehicles and smaller humanoid bots. These agent types are defined in the lumberyard_version\dev\your_project_name\Scripts\AI\Navigation.xml file.

To define an agent type on your navigation area, see the Navigation Area (p. 693) component.

Agent Speed

Sets the speed of the agent while navigating when using the Transform or Physics movement methods.

Default value: 1

Agent Radius

Sets the entity radius for navigation purposes. Independent of physics or other collision concerns, the pathfinder uses this value to move around an area with obstacles while cutting corners.

Default value: 4

Arrival Distance Threshold

Sets the minimum distance from an end point when an entity's movement stops and is considered complete.

Default value: 0.25

Repath Threshold

Sets the minimum distance from the previously known location before an entity's new path is calculated.

Default value: 1

Movement Method

Sets the movement method to use when following a path. This can be Transform, Physics, or Custom.
Default value: Transform

- **Transform** – Move the entity that this component is on using the Transform bus. This method ignores all physics so the object may go through walls and terrain.

- **Physics** – Move the entity using physics if the entity has a PhysX Rigid Body, PhysX Character Controller, Rigid Body Physics or Character Physics component. If the entity does not have one of these valid physics components it will not move.

- **Custom** – Provide path updates and let the game logic move the entity however they want. This method is useful when you want to move an animated entity that uses root motion. By listening to the `OnTraversalPathUpdate` notification, you can move your entity toward the next point along a path. Once the entity gets within the arrival distance threshold another `OnTraversalPathUpdate` notification with the next path position will be provided and so on until the end of the path is reached.

**Allow Vertical Navigation**

Set to true if you want to allow the navigation agent to include the vertical velocity when navigating a path, or false if you just want the velocity to be constrained to the X and Y plane (2D). Vertical navigation can be used for flying entities or entities that move with the Transform method but must move vertically. Enabling this property can also help prevent "stair stepping" for entities moving down ramps or steep terrain.

Default value: `false`

**NavigationComponentRequestBus EBus Interface**

Use the following request functions with the `NavigationComponentRequestBus` event bus (EBus) interface to communicate with other components of your game.

For more information about using the EBus interface, see *Working with the Event Bus (EBus) system* (p. 1851).

**FindPath**

Finds a requested path configuration.

**Parameters**

- `request` – Allows the issuer of the request to override one, all, or none of the pathfinding configuration defaults for this entity.

**Return**

A unique identifier for this pathfinding request.

**Scriptable**

- No

**FindPathToEntity**

Creates a pathfinding request to navigate toward the specified entity.

**Parameters**

- `EntityId` – ID of the entity toward which you want to navigate.

**Return**

A unique identifier for the pathfinding request.
**Scriptable**

Yes

**FindPathToPosition**

Creates a pathfinding request to navigate towards the specified world position. Note that while this may seem like the obvious simple choice for pathing, it is often more useful to use FindPathToEntity with a dummy entity because then the pathing will automatically update if you move the dummy entity to a new location before pathing is complete.

**Parameters**

- **Destination** – World position you want to navigate to.

**Return**

- A unique identifier for the pathfinding request.

**Scriptable**

Yes

**Stop**

Stops all pathfinding operations for the provided requestId. Use the ID to ensure that the request you want to cancel is the request that is currently processing. If the specified requestId is different from the ID of the current request, then the Stop command is ignored.

**Parameters**

- **requestId** – ID of the request to cancel.

**Return**

- None

**Scriptable**

Yes

**GetAgentSpeed**

Returns the current AI Agent's speed.

**Parameters**

- None

**Return**

- Returns the current agent's speed as a float.

**Scriptable**

Yes

**SetAgentSpeed**

Updates the AI Agent's speed.
Parameters

agentSpeed – Specifies the new agent speed as a float.

Return

None

Scriptable

Yes

GetAgentMovementMethod

Returns the current AI Agent's movement method.

Parameters

None

Return

Returns the current agent's movement method.

Scriptable

Yes

SetAgentMovementMethod

Updates the AI Agent's movement method.

Parameters

movementMethod – Specifies the new agent movement method (Transform, Physics or Custom).

Return

None

Scriptable

Yes

NavigationComponentNotificationBus EBus Interface

Use the following notification functions with the NavigationComponentNotificationBus event bus (EBus) interface to communicate with other components of your game.

For more information about using the EBus interface, see Working with the Event Bus (EBus) system (p. 1851).

OnSearchingForPath

Indicates that the pathfinding request has been submitted to the navigation system.

Parameters

requestId – ID of the request for which the path is being searched.

Return

None
Scriptable
  Yes

**OnPathFound**
Indicates that a path has been found for the indicated request.

**Parameters**

  - requestID – ID of the request for which the path has been found.
  - currentPath – The path calculated by the pathfinder.

**Return**
Returns a boolean value indicating whether this path is to be traversed.

Scriptable
  No

**OnTraversalStarted**
Indicates that traversal for the indicated request has started.

**Parameters**

  - requestId – ID of the request for which traversal has started.

**Return**
None

Scriptable
  Yes

**OnTraversalInProgress**
Indicates that traversal for the indicated request is in progress.

**Parameters**

  - requestId – ID of the request for which traversal is in progress.
  - distanceRemaining – Remaining distance in this path.

**Return**
None

Scriptable
  Yes

**OnTraversalPathUpdate**
Indicates that the path for the traversal has updated. If the `nextPathPosition` and `inflectionPosition` are equal, they represent the end of the path.
Parameters

requestId – ID of the request for which traversal is in progress.

nextPathPosition – Furthest point on the path we can move to without colliding with anything.

inflectionPosition – Next point on the path beyond nextPathPosition that deviates from a straight-line path.

Return

None

Scriptable

Yes

OnTraversalComplete

Indicates that traversal for the indicated request completed successfully.

Parameters

requestId – ID of the request for which traversal has completed.

Return

None

Scriptable

Yes

OnTraversalCancelled

Indicates that traversal for the indicated request was canceled before successful completion. A path request may be canceled if no path could be found or if the request was stopped by the game.

Parameters

requestId – ID of the request for which traversal was canceled.

Return

None

Scriptable

Yes

Navigation Pathing Cvars

ai_DrawPathFollower

Enables PathFollower debug drawing, displaying agent paths and safe follow target.

0 – Off
1 – On
Navigation Area

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Navigation Area component defines the features of a navigable area or volume for use by the AI System. You use this component with the Polygon Prism Shape (p. 708) component, which defines the volume of the navigation area.

**Note**
When you add a Navigation Area component, you must also add the Polygon Prism Shape component.

For instructions on how to adjust the Polygon Prism Shape component, see Polygon Prism Shape (p. 708).

When you create a Navigation Area, all areas that are traversable by the specified AI Agent Type (p. 696) show as blue when you render the navigation mesh (p. 695). Any areas that are not traversable render as blank areas, such as deep pits and steep slopes. Exclusion areas and areas around static objects also render as blank areas. Even if your navigation area is divided into separate pieces by static objects, terrain features, and exclusion areas, each traversable area renders blue.
You can use a **Navigation Seed (p. 699)** component to fine-tune AI accessibility.

### To add a Navigation Area

1. In the Viewport, near where you want to create your navigation area, right-click and choose **Create entity**.
2. With your new entity selected in the **Entity Outliner**, add (p. 479) the **Navigation Area** component to it.

![Navigation Area component](image)

3. In the **Navigation Area** component, next to **Agent Types**, click +.

   Next to [0], select **MediumSizedCharacters**. This property defines the **types of agents (p. 696)** that can navigate in this area.

4. Add the **Polygon Prism** component. Adjust the size and shape (p. 710) of the **Polygon Prism**. Ensure that your terrain and objects intersect with the volume of the polygon prism. Adjust the **height (p. 711)** if necessary.

   If your polygon prism hovers above your terrain and does not fully intersect with it, the navigation system does not produce the appropriate traversable areas. The following examples show a navigation area that is too high above the terrain (1), and a navigation area appropriately situated on the terrain (2). If your navigation area is too high, use the **move (p. 201)** tool to lower the Z (up and down) position of the entity.

![Navigation examples](image)

### To view the generated Navigation Area mesh

1. In Lumberyard Editor, choose **Game, AI, Show Navigation Areas**.
2. In Lumberyard Editor, choose **Game, AI, View Agent Type**, and then enable the agent type that you want to display.
3. In Lumberyard Editor, choose **Game, AI, Continuous Update** to show changes in the navigation mesh as you modify the terrain or area.
Navigation Area

A navigation mesh shows traversable areas in blue.

**Topics**
- Navigation Area Component Properties (p. 696)
- Navigating Around Static Objects (p. 696)
- Creating Navigation Mesh Exclusion Areas (p. 697)
- Navigation Physics Integration (p. 698)
- NavigationAreaRequestBus EBus Interface (p. 698)

**Navigation Area Component Properties**

The **Navigation Area** component has the following properties:

**Agent Types**

Specifies the types of AI that can traverse this navigation area. These agent types are defined in `lumberyard_version\dev\your_project_name\Scripts\AI\Navigation.xml`. To specify multiple agent types for this area, click the + icon.

You use this property to restrict which agents can navigate within that area. For example, you can allow characters to navigate within a narrow corridor but restrict vehicles.

To define an agent type on your AI, see the **Navigation** (p. 686) component.

**Exclusion**

When selected, creates a subtractive navigation area. This creates a cutout within an existing navigation mesh. For more information, see **Creating Navigation Mesh Exclusion Areas** (p. 697).

**Navigating Around Static Objects**

When Lumberyard creates the navigation mesh, it can automatically exclude areas that should not be traversable, such as a large boulder or tree trunk. To ensure that these areas are correctly detected by the navigation system, you must specify its **Transform** component as static.

**To mark an entity as static**

1. In the **Entity Outliner**, select the entity. This can be a tree, boulder, building, or any object that you don't want the AI to walk through.
2. In the **Transform (p. 878)** component, select the **Static** property.

The following example shows a navigation mesh around a boulder when the **Static** property is not selected.

![Image 1](image1.png)

The following example shows the same navigation mesh, but with the **Static** property selected on the boulder.

![Image 2](image2.png)

### Creating Navigation Mesh Exclusion Areas

You can use the **Navigation Area** component to manually create areas to exclude from the navigation mesh. This means that the AI agents cannot traverse these areas. To do this, you create a navigation area and select the **Exclusion** property, as shown in the following image.

![Image 3](image3.png)

The following example shows a navigation mesh (1) and the same navigation mesh with an exclusion area (2).
To create an exclusion area

1. If you have not already, you must first create a navigation area (p. 695) that is not an exclusion area.
2. Create an entity, and add to it the **Navigation Area** and **Polygon Prism Shape** components.
3. In the **Navigation Area** component, select the **Exclusion** property.
4. In the **Viewport**, place the exclusion area within the navigation mesh.
5. Shape the polygon to the preferred shape for the exclusion area.

**Navigation Physics Integration**

The navigation system builds the navigation mesh based on all the static physics colliders provided by the physics system, including terrain. By default, both the CryPhysics and PhysX systems (AZ::Physics) are supported. This can be changed with the *ai_NavPhysicsMode* cvar:

**ai_NavPhysicsMode**

Navigation physics integration mode which determines where collider and terrain data used in navigation mesh calculations comes from.

Default: 1

- 0 – CryPhysics only
- 1 – CryPhysics and AZ::Physics
- 2 – AZ::Physics only

**Physics integration details**

When AZ::Physics integration mode is enabled, the navigation mesh voxelizer issues a WorldRequestBus::Overlap static query to gather colliders within a bounding box. Shape geometry is returned from the new AZ::Shape::GetGeometry() method. PhysX colliders for terrain, shapes, and meshes will provide triangle data in an operation that acquires a PhysX scene read lock while retrieving geometry data.

**NavigationAreaRequestBus EBus Interface**

Use the following request function with the NavigationAreaRequestBus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851)
**RefreshArea**

You can use the `PolygonPrismShapeComponentRequestBus` (p. 716) to modify the polygon prism area by adding, removing, and updating its vertex positions. Use `RefreshArea` to update the navigation area after making changes to the area.

**Parameters**

None

**Return**

`AZ::ConstPolygonPrismPtr`

**Scriptable**

No

**Note**

The Navigation Area component depends on the Polygon Prism component, which also uses `VertexContainer` functions. For more information, see Vertex Containers (p. 904).

---

**Navigation Seed**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

The **Navigation Seed** component marks chunks of the **Navigation Area** (p. 693) that are accessible to AI agents. Game developers can use this component as a visual aid to determine where AI agents can go.

The **Navigation Area** (p. 693) component can generate a complex-looking mesh with disconnected islands. If this happens, it can be difficult to determine the precise places that your AI can reach. In this case, use the **Navigation Seed** component to render a color-coded map. The blue chunks are accessible to AI and the red chunks are inaccessible.
For example, static objects (p. 696), exclusion areas (p. 697), or terrain features can divide a navigation area (p. 693) into multiple chunks. The Navigation Seed component marks in blue where AI can reach if they are already in that chunk (for example, if they spawned there). You might have multiple navigation areas in one location, such as for different agent types (p. 687). In this case, use the Navigation Seed component to calculate accessibility for all agent types (p. 696) or a specific agent type in the chunk where you placed the navigation seed.

**To use the Navigation Seed component**

1. Create a navigation area (p. 693).
2. Divide the navigation area into multiple chunks using static objects (p. 696), exclusion areas (p. 697), or terrain.
3. Add (p. 479) the Navigation Seed component to the navigation area entity or to a separate entity.
4. If you want to specify an agent type (p. 687), select it in the Navigation Seed component.
5. Move (p. 201) the seed around.

   If you turned on visualization (p. 695), all AI-accessible areas render blue in the chunk where you placed the navigation seed. Inaccessible areas render red.

**To calculate accessibility for agent types**

- In the Navigation Seed component, do one of the following:
  
  a. To calculate accessibility for all agent types, leave the Agent Type field blank.
  b. To calculate accessibility for a specific agent type, select a type in the Agent Type drop-down list.

By default, the navigation seed visualization system is not enabled. You must use the console to enable some flags.

**To enable Navigation Seed visualization**

- Enable the following console variables. To do this, set the value to 1.
  
  ai_MNMDebugAccessibility (In Lumberyard Editor, you can also choose Game, AI, Visualize Navigation Accessibility.)
  
  ai_DebugDraw
  
  ai_DebugDrawNavigation

For more information, see Using the Console Window (p. 210).

**Note**

The Navigation Seed component exists only in Lumberyard Editor.

---

**Network Binding**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Network Binding component marks an entity as able to be replicated across the network. It interfaces with the NetBindable interface in order to call the functions specified there.
OccluderArea

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the OccluderArea component to create a custom-shaped occlusion plane with four vertices. This is useful if you don't want Lumberyard to render everything that is behind the OccluderArea component. This can result in better performance in areas where automatic occlusion doesn't work well. For example, if you have many objects behind a wall, you can add an occluder area behind the wall so that those objects don't appear.

Note
You can't modify the OccluderArea (p. 701), Portal (p. 717), and VisArea (p. 918) components at runtime.

Topics
• OccluderArea Component Properties (p. 701)
• Occluder Area Component Example (p. 702)

OccluderArea Component Properties

The OccluderArea component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplayFilled</td>
<td>Displays the occluder area as a filled volume in the Lumberyard Editor viewport.</td>
</tr>
<tr>
<td>CullDistRatio</td>
<td>A multiplier on the range where the culling effect stops.</td>
</tr>
<tr>
<td>UseInIndoors</td>
<td>The occluder area works inside visible areas.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DoubleSide</td>
<td>Specifies whether the occluder area works from both sides.</td>
</tr>
<tr>
<td>Vertices</td>
<td>The vertices that define the shape of the occluder area. Occluder areas always have four vertices.</td>
</tr>
<tr>
<td>Edit</td>
<td>Choose Edit, and the component is locked for editing. For more information, see Editing Components in the Viewport (p. 483).</td>
</tr>
</tbody>
</table>

**Occluder Area Component Example**

The following example shows two boxes. One box is hidden behind the occluder area, but you can see its shadow. The other box is outside the occluder area and appears normally on screen.

**Particle**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Particle** component to place a single particle emitter on an entity. An entity can have multiple **Particle** components.
Particle Component Properties

The Particle component has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>If set, renders the emitter.</td>
</tr>
<tr>
<td>Enable</td>
<td>If set, activates the particle effect.</td>
</tr>
<tr>
<td>Particle effect library</td>
<td>Specifies the particle effect library.</td>
</tr>
<tr>
<td></td>
<td>The Particle component uses the effects from an XML file.</td>
</tr>
<tr>
<td>Emitters</td>
<td>Select an emitter from the list after specifying a particle effect library.</td>
</tr>
</tbody>
</table>

Version 1.28
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If emitters are not listed, you can create emitters in the [Particle Editor](p. 1504).</td>
</tr>
<tr>
<td>Color tint</td>
<td>Select the color of the effect with the color swatch.</td>
</tr>
<tr>
<td>Pre-Roll</td>
<td>If set, the emitter behaves as though it has been running indefinitely.</td>
</tr>
<tr>
<td>Count scale</td>
<td>Sets the multiplier for the particle count.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1000</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Time scale</td>
<td>Sets the multiplier for the emitter time evolution.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1000</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Pulse period</td>
<td>Sets the frequency at which to restart the emitter.</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Global size scale</td>
<td>Sets the multiplier for all effect sizes.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 100</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Particle size scale x</td>
<td>Sets the multiplier for the particle size on the x-axis.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 100</td>
</tr>
<tr>
<td>Particle size scale y</td>
<td>Sets the multiplier for the particle size on the y-axis.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 100</td>
</tr>
<tr>
<td>Particle size scale z</td>
<td>Sets the multiplier for the particle size on the z-axis. This parameter applies to geometry particles only. For more information, see the Geometry (p. 1539) parameter.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 100</td>
</tr>
<tr>
<td>Particle size scale random</td>
<td>Randomizes the particle size scale.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1</td>
</tr>
<tr>
<td>Speed scale</td>
<td>Sets the multiplier for the particle emission speed.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 1000</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Strength Curve Time     | Controls all **Strength Over Emitter Life** curves. The curves will use this **Strength Curve Time** parameter instead of the actual emitter lifetime.  
Negative values are ignored.  
Valid values: \(-1 \text{ to } 1\) |
| Ignore rotation         | If set, ignores the entity's rotation.                                      |
| Not attached            | If set, ignores the entity's position. The emitter does not follow its entity. |
| Register by bounding box| If set, uses the bounding box instead of the entity's position to appear in the visible area. |
| Use LOD                 | If cleared, ignores the emitter's level of detail (LOD).                    |
| Target Entity           | Uses the specified target entity for emitters with the **Target Attraction** parameter or similar features enabled.  
For more information, see **Movement Attribute** (p. 1529). |
| View distance multiplier| Adjusts the maximum view distance. If the value is 1, the default is used. If the value is 1.1, the view distance would be 10% further than the default.  
Set the value to 100 for infinite visibility.  
Valid values: 0 to 100 |
| Use VisAreas            | Allow visible areas to control this component's visibility.                |
| Enable audio            | If set, enables audio.                                                     |
| Audio RTPC              | Select the audio **Real-Time Parameter Control (RTPC)** (p. 3147) that the particle effect instance drives. |

### EBus Request Bus Interface

Use the following request function with the EBus (event bus) interface to communicate with other components of your game.

For more information, see **Working with the Event Bus (EBus) system** (p. 1851).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide</td>
<td>Hides the emitter.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Show</td>
<td>Shows the emitter.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Point Light**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the **Point Light** component on an entity to create a point of light.

The **Point Light** component has the following settings:

**Visible**

- Shows the light.

**On initially**

- Specify if the light is on when created.

**General Settings**

See the following general settings:

**Color**

- The color of the light.
  - Default value: 0xFFFFFFFF

**Diffuse multiplier**

- Sets the strength of the diffuse color.
  - Default value: 1

**Specular multiplier**

- Sets the strength of the specular brightness.
  - Default value: 1

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetVisibility</td>
<td>Sets an explicit value for emitter visibility.</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

The following script uses the EBus interface.

```plaintext
function example:OnActivate()
    ParticleComponentRequestBus.Event.Show(self.entityId)
    ParticleComponentRequestBus.Event.Hide(self.entityId)
    ParticleComponentRequestBus.Event.SetVisibility(self.entityId, false)
end
```
Ambient

Light acts as a multiplier for cubemap values.

Point Light Settings

See the following point light settings:

Max Distance

Maximum distance, in a radius, from which this light is visible.

Default value: 2

Attenuation bulb size

Radius in meters before light falloff begins.

Default value: 0

Options

See the following options:

View distance multiplier

Adjusts the maximum view distance. For example, 1.0 uses the default and 1.1 is 10% farther than the default.

Default value: 1

Minimum spec

The minimum specification value at which this light is enabled.

Default value: Low

Cast shadow spec

The minimum specification at which shadows are cast.

Default value: Never

Voxel GI mode

Mode for light interaction with voxel global illumination (GI). Choose None, Static, or Dynamic.

Use VisAreas

Light is affected by visible areas. If unselected, light ignores visible areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.
Volumetric fog

Light affects volumetric fog and surrounding area.

Shadow Settings

See the following shadow settings:

Terrain Shadows

Includes the terrain in the shadow casters for this light.

Animation

See the following animation settings:

Style

Enter a number to specify a preset light animation curve to play as defined in the Light.cfx file. Valid values are 0 to 48. You can also use values 40 to 48 for testing and debugging.

Default value: 0

Speed

Multiple of the base animation rate. For example, a value of 2.0 makes an animation play twice as fast.

Default value: 1

Phase

Animation start offset from 0 to 1. A value of 0.1 is 10% into the animation. For example, you can use this setting to prevent lights in the same scene, with the same animation, from being animated in unison.

Default value: 0

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

All light components share common EBus functions. For more information, see Light Components EBus Request Bus Interface (p. 548).

Polygon Prism Shape

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use the **Polygon Prism Shape** component to define a volume. A right polygonal prism is a 3D prism made from two translated polygons connected by rectangles. The **Polygon Prism Shape** component is defined by one polygon and a height property. All vertices lie on the local plane, where $z = 0$. The polygon is defined in the xy-plane of the entity. All vertices are in the local space of the entity. You can use the **Polygon Prism Shape** component with the **Trigger Area** (p. 890) component to create a volume for the trigger area bounds.

**Note**
The **Polygon Prism Shape** component supports only simple polygons. It can't have self-intersecting sides. Parallelogram sides aren't supported.

**Contents**
- Polygon Prism Shape Component Properties (p. 709)
- Working with Polygon Prism Components (p. 710)
- EBus Request Bus Interface (p. 716)
  - Example Polygon Prism Component EBus Request (p. 717)

### Polygon Prism Shape Component Properties

The **Polygon Prism Shape** component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the shape in game mode in Lumberyard Editor. To enter game mode, press Ctrl + G.</td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the polygon prism shape, in meters.</td>
</tr>
<tr>
<td>Edit</td>
<td>Choose <strong>Edit</strong>, and the component is locked for editing. For more information, see Editing Components in the Viewport (p. 483).</td>
</tr>
</tbody>
</table>

By default, when you add the **Polygon Prism Shape** component to an entity, the component has four vertices.
Working with Polygon Prism Components

Each vertex has one planar and two linear manipulators. Combined, these are called the translation manipulator. You can use the translation manipulator to move the vertices. The translation manipulator limits moving a vertex only to the xy-plane. No surface manipulator is present as all vertices must exist on the same plane. You can't change the z-position of the polygon in the local space.

To change the vertex positions

- Do one of the following:
  - In the Polygon Prism Shape component property window, change the vertex values.
  - Select a vertex. In the translation manipulator, click and drag the green or red arrow to a position.
Note
The manipulators follow the grid snap setting that you can configure in the Lumberyard Editor toolbar.

You can use the blue arrow in the center of the volume to change the height.

To change the height of the polygon prism
- Drag the blue arrow to change the height of the polygon prism.
Note

If the entity is scaled, the manipulators continue to behave correctly, but the scaling must be uniform so that the x, y, and z scale values match.

To select multiple vertices

- Press *Shift* and select the vertices. Selected vertices appear yellow.

Note

Currently, you can’t click and draw a box around the vertices to select them. You can also snap a vertex to another position on the terrain.
To snap a vertex to another position

1. Select a vertex.
2. Hold **Shift+Ctrl** and click another position on the terrain. The vertex snaps to that position.

You can also add vertices to the polygon prism.

To add a vertex to the polygon prism

1. Pause on a line and hold **Ctrl**. A preview appears where you can add a vertex.
2. Click to add the vertex to the polygon prism.
You can also delete vertices from the polygon prism.

To delete a vertex from a polygon prism

1. Hold Alt and pause on a vertex. The vertex appears gray.
2. Click the vertex to remove it from the polygon prism. You can also select a vertex and press **Delete**.
**EBus Request Bus Interface**

Use the following request functions with the `PolygonPrismShapeComponentRequestBus` EBus interface to communicate with other components of your game. The **Polygon Prism Shape** component also uses `VertexContainer` functions. For more information, see [Vertex Containers](p. 904).

For more information about using the event bus (EBus) interface, see [Working with the Event Bus (EBus) system](p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPolygonPrism</td>
<td>Returns a constant pointer to the underlying polygon prism shape data.</td>
<td>None</td>
<td><code>AZ::ConstPolygonPrismPtr</code></td>
<td>Yes</td>
</tr>
<tr>
<td>SetHeight</td>
<td>Sets the height of the polygon prism shape.</td>
<td>Height</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Default value: 1.0

Min: 0.0

Max: N/A
Example Polygon Prism Component EBus Request

The following script uses the request bus interface.

```lua
-- PolygonPrism Interface
local polygonPrism = PolygonPrismShapeComponentRequestBus.Event.GetPolygonPrism(self.entityId);

polygonPrism.height
polygonPrism.vertexContainer:Size()
polygonPrism.vertexContainer[1]

PolygonPrismShapeComponentRequestBus.Event.SetHeight(self.entityId, 5.0);

-- VertexContainer interface
local firstVertex = spline.vertexContainer[1];
local lastVertex = spline.vertexContainer[spline.vertexContainer:Size()];

-- Attention: You can use the bus interface directly, but here (at the moment at least) indexing will start from 0, not 1 as is the norm in Lua. Instead, you should use the functions directly on the VertexContainer, listed below
PolygonPrismShapeComponentRequestBus.Event.AddVertex(self.entityId, Vector2(10, 10));
PolygonPrismShapeComponentRequestBus.Event.UpdateVertex(self.entityId, 0, firstVertex + Vector2(5, 5));
PolygonPrismShapeComponentRequestBus.Event.InsertVertex(self.entityId, spline.vertexContainer:Size() - 1, lastVertex);
PolygonPrismShapeComponentRequestBus.Event.ClearVertices(self.entityId);
PolygonPrismShapeComponentRequestBus.Event.RemoveVertex(self.entityId, spline.vertexContainer:Size() - 1);

-- Prefer these functions - indexing will start from 1
polygonPrism.vertexContainer:AddVertex(lastVertex + Vector2(5, 5));
polygonPrism.vertexContainer:UpdateVertex(1, firstVertex + Vector2(10, 10));
polygonPrism.vertexContainer:InsertVertex(polygonPrism.vertexContainer:Size(), lastVertex + Vector2(2, 2));
polygonPrism.vertexContainer:Clear();
polygonPrism.vertexContainer:RemoveVertex(polygonPrism.vertexContainer:Size());
```

Portal

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Portal component to create efficient indoor areas with the VisArea (p. 918) component. You can use portals to specify areas where you can look out of a visible area and see beyond it, such as windows or doors between visible areas.

To use the Portal component, its volume must overlap at least two or more visible areas.

Note
You can't modify the OccluderArea (p. 701), Portal (p. 717), and VisArea (p. 918) components at runtime.

Topics
Portal Component Properties

The Portal component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>The height of the portal.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Displays the portal as a filled volume in the Lumberyard Editor viewport.</td>
</tr>
<tr>
<td>AffectedBySun</td>
<td>Specifies whether the sun affects objects in the portal.</td>
</tr>
<tr>
<td>ViewDistRatio</td>
<td>A multiplier on how far the portal is rendered.</td>
</tr>
<tr>
<td>SkyOnly</td>
<td>Renders only the skybox outside the portal.</td>
</tr>
<tr>
<td>OceanIsVisible</td>
<td>Renders the ocean outside the portal.</td>
</tr>
<tr>
<td>UseDeepness</td>
<td>The portal is treated as an object with a volume instead of a plane.</td>
</tr>
<tr>
<td></td>
<td>If false, the portal acts as if it does not have a volume and is</td>
</tr>
<tr>
<td></td>
<td>treated like a plane between the four floor points.</td>
</tr>
<tr>
<td>DoubleSide</td>
<td>Cameras can look through the portal from both sides. If false, the portal</td>
</tr>
<tr>
<td></td>
<td>only works from one side.</td>
</tr>
<tr>
<td>LightBlending</td>
<td>Specify whether to blend light between connected visible areas.</td>
</tr>
</tbody>
</table>
### Portal Component Examples

#### Example 1

In the following examples, two boxes are inside several visible areas. The visible areas are connected by the portals, which are green. The portal lets you see between the visible areas. The shadows of the boxes inside the visible areas appear because the **AffectedBySun** option is enabled.

![Portal Example 1](image)

#### Example 2

The following example shows how the portal lets you see between two visible areas. Although the boxes are in separate visible areas, the portal creates a window between them so that you can see both boxes.

![Portal Example 2](image)
Example 3

In the following example, you see the portal at an angle. You can't see the box in the other visible area because there is no portal in that line of sight.
Example 4

In the following example, you can see a box through a different portal, but because the DoubleSide option is disabled, the portal is one-sided.
Example 5

In the following example, if you go through the portal and look back, you can see the other visible areas, but you can't see the boxes in those areas. You can't see through portals that aren't connected to the visible area that you're in.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the **Projector Light** component on an entity to project a light.

The **Projector Light** component has the following settings:

**Visible**

Show the light.

**On initially**

Specify if the light is on when created.

**General Settings**

See the following general settings:

**Color**

The color of the light.
Default value: 0xFFFFFFFF

**Diffuse multiplier**

Sets the strength of the diffuse color.

Default value: 1

**Specular multiplier**

Sets the strength of the specular brightness.

Default value: 1

**Ambient**

If selected, light acts as a multiplier for cubemap values.

---

**Project Light Settings**

See the following point light settings:

**Max Distance**

Maximum distance that the projector light extends, in meters.

Default value: 5

**Attenuation bulb size**

Radius before light begins to fade, in meters.

Default value: 0

**FOV**

Projector light's field of view (FOV), in degrees.

Default value: 90

**Near plane**

Distance of the near plane to the projector light, in meters. Objects behind this plane are not affected by the projector light.

**Texture**

Projector light's texture file. Without a texture, light does not shine.

**Material**

Projector light's material file.

---

**Options**

See the following options:

**View distance multiplier**

Adjusts the maximum view distance. For example, 1.0 uses the default and 1.1 is 10% farther than the default.
Default value: 1

**Minimum spec**

The minimum specification value at which this light is enabled.

Default value: Low

**Cast shadow spec**

The minimum specification at which shadows are cast.

Default value: Never

**Voxel GI mode**

Mode for light interaction with voxel global illumination (GI). Choose None, Static, or Dynamic.

**Use VisAreas**

Light is affected by visible areas. If unselected, light ignores visible areas.

**Indoor only**

Light is only rendered indoors.

**Affects this area only**

Light affects only the immediate area.

**Volumetric fog only**

Light affects only volumetric fog.

**Volumetric fog**

Light affects volumetric fog and surrounding area.

### Shadow Settings

See the following shadow settings:

**Terrain Shadows**

Includes the terrain in the shadow casters for this light.

### Animation

See the following animation settings:

**Style**

Enter a number to specify a preset light animation curve to play as defined in the `Light.cfx` file. Valid values are 0 to 48. You can also use values 40 to 48 for testing and debugging.

Default value: 0

**Speed**

Multiple of the base animation rate. For example, a value of 2.0 causes an animation to play twice as fast.
Default value: 1

**Phase**

Animation start offset from 0 to 1. A value of 0.1 is 10% into the animation.

For example, you can use this setting to prevent lights in the same scene, with the same animation, from being animated in unison.

Default value: 0

**Additional Resources**

- For more information about the **Projector Light** component, see the following:
  - Intro to Environment Lighting
  - Getting Started: Using Lighting Tutorial (video)

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

All light components share common EBus functions. For more information, see Light Components EBus Request Bus Interface (p. 548).

**PhysX Ball Joint component**

With the **PhysX Ball Joint** component, you can create a dynamic ball joint that constrains an entity to the joint with freedom to rotate around the y- and z-axes of the joint.
**PhysX Ball Joint component properties**

![PhysX Ball Joint component properties](image)

- **Local Position**
  - Specify the position of the joint relative to the entity transform.

- **Local Rotation**
  - Specify the rotation of the joint relative to the entity transform.

- **Lead Entity**
  - Specify the parent entity that will drive the joint.

- **Breakable**
  - When enabled, the joint will break if sufficient force is applied. Enabling **Breakable** exposes the **Maximum Force** and **Maximum Torque** properties.

**Note**

- **PhysX Rigid Body** components that have their **Compute Mass** property enabled might have very large mass values. If the entity containing the joint component or the leader entity have their **Compute Mass** property enabled, the **Maximum Force** and **Maximum Torque** properties might require very high values to resist breaking.
**Maximum Force**

When **Breakable** is enabled, specify the maximum force the joint can sustain before breaking. Valid values range from 0.01 to Infinity.

**Maximum Torque**

When **Breakable** is enabled, specify the maximum torque the joint can sustain before breaking. Valid values range from 0.01 to Infinity.

**Display Setup in Viewport**

When enabled, a cone that represents the orientation and limits of the ball joint and a line that represents the connection between the joint and its leader are displayed. The shape of the cone is defined by the **Y axis angular limit** and the **Z axis angular limit**.

**Select Lead on Snap**

When enabled, snapping the joint to an entity in component mode will set the entity as the **Lead Entity**. The entity containing the joint component is excluded from this operation.

**Lead-Follower Collide**

When enabled, the lead entity and follower entity (the entity containing the joint component) will collide.

**Limit**

When enabled, the lead entity's movement around the joint axes is constrained by angular limits. Enabling **Limit** exposes the **Soft Limit**, the **Y axis angular limit**, and the **Z axis angular limit** properties.

**Soft Limit**

When enabled, the lead entity's movement around the joint axes is allowed to pass the specified limit. With **Soft Limit** enabled, when the lead entity rotates past the limit, the lead entity's movement is treated as a spring and will slow, then spring back to the limit area. Enabling **Soft Limit** exposes the **Damping** and **Stiffness** properties.

**Damping**

When **Soft Limit** is enabled, the spring's drive relative to the velocity of the follower when outside the rotation limit. Valid values range from 0.001 to 1000000.0.

**Stiffness**

When **Soft Limit** is enabled, the spring's drive relative to the position of the follower when outside the rotation limit. Valid values range from 0.001 to 1000000.0.

**Y axis angular limit**

The rotation limit around the joint's y-axis when **Limit** is enabled. Valid values range from 0.1 to 180.0.

**Z axis angular limit**

The rotation limit around the joint's z-axis when **Limit** is enabled. Valid values range from 0.1 to 180.0.

**Edit**

When clicked, component edit mode is enabled. In component edit mode, all components are locked except for the **PhysX Ball Joint** component. The properties of the **PhysX Ball Joint** component can be edited in **Perspective**. Press **Tab** to cycle through the component edit modes. Click **Done** to exit component mode.
PhysX Character Controller

You can use the PhysX Character Controller component to implement basic character interactions with the physical world. For example, you can prevent characters from walking through walls or passing through terrain. You can also control interactions with slopes and steps and manage interactions with other characters.

The following diagram shows some of the features of the PhysX Character Controller component. Because it is usually more convenient to work with a character's foot position, the entity position coincides with the base of the controller. For details about the contact offset, see Contact Offset (p. 733).

The PhysX Character Controller component requires the PhysX Characters (p. 1188) gem, which you can enable (p. 1064) in the Project Configurator.

Topics

- Using the PhysX Character Controller Component (p. 730)
- PhysX Character Controller Properties (p. 730)
- Differences Between PhysX and Legacy Character Physics Components (p. 734)
Using the PhysX Character Controller Component

To use the **PhysX Character Controller** component, add it to an entity (p. 479) that represents a character.

You can control the character's movement by using Script Canvas, the C++ API, or animation systems that use the C++ API.

**PhysX Character Controller Properties**

You can configure the properties for the **PhysX Character Controller** component in the **Entity Inspector (p. 475)**.

The **PhysX Character Controller** has the following component properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision Layer</td>
<td>Collision layer assigned to the controller. The default is <strong>Default</strong>.</td>
</tr>
<tr>
<td>Collides With</td>
<td>Collision layers that this character controller collides with. Possible</td>
</tr>
<tr>
<td></td>
<td>values are those that you define in the collision groups section of the</td>
</tr>
<tr>
<td></td>
<td>PhysX configuration.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>You can specify the following values:</td>
</tr>
<tr>
<td></td>
<td>• All</td>
</tr>
<tr>
<td></td>
<td>• None</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The collision filters determine whether dynamic objects collide with the controller. A separate set of filters control what objects can impede the character from moving. The movement filters are currently hard-coded so that static objects obstruct character movement.</td>
</tr>
<tr>
<td>Material Library</td>
<td>PhysX material (p. 2797) assigned to this character controller.</td>
</tr>
<tr>
<td></td>
<td>• Click the browse icon (...) to open the Pick Physics Material dialog box.</td>
</tr>
<tr>
<td></td>
<td>• Click the Open in Asset Editor icon to open the material in the Asset Editor.</td>
</tr>
<tr>
<td></td>
<td>• Click the X icon to clear the asset.</td>
</tr>
<tr>
<td>Material Slot</td>
<td>Specifies a material from the material library. To select a material from the library, click the drop-down list. The default is the first value configured.</td>
</tr>
<tr>
<td></td>
<td>If the material library is empty, this option does not appear and the default material values (0.5 and average for all parameters) are used.</td>
</tr>
<tr>
<td>Maximum Slope Angle (p. 732)</td>
<td>Angle in degrees of the largest slope that the character controller can climb.</td>
</tr>
<tr>
<td>Step Height (p. 732)</td>
<td>Height of steps in meters that the character controller can traverse.</td>
</tr>
<tr>
<td>Minimum Movement Distance</td>
<td>Distance in meters below which the controller doesn't attempt to move. Used to avoid jittering.</td>
</tr>
<tr>
<td>Collider Tag</td>
<td>A tag string used to identify the collider associated with the character controller.</td>
</tr>
<tr>
<td>Slope Behavior</td>
<td>Behavior of the controller on surfaces above the maximum slope.</td>
</tr>
<tr>
<td></td>
<td>You can specify the following values:</td>
</tr>
<tr>
<td></td>
<td>• Prevent Climbing</td>
</tr>
<tr>
<td></td>
<td>• Force Sliding</td>
</tr>
<tr>
<td></td>
<td>The default is Prevent Climbing.</td>
</tr>
<tr>
<td>Contact Offset (p. 733)</td>
<td>Additional distance in meters beyond the controller that is monitored for potential contact. Used for smoother contact resolution.</td>
</tr>
<tr>
<td>Scale</td>
<td>Scales the size of the collider created in PhysX relative to the dimensions specified for the controller. A value slightly smaller than 1 is recommended.</td>
</tr>
<tr>
<td></td>
<td>The default is 0.8.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **Shape (p. 734)**| Shape of the character controller. You can specify the following values:  
• Box  
• Capsule  
The default is **Capsule**. |
| **Height (Capsule Only)** | Height of the capsule in meters. |
| **Radius (Capsule Only)** | Radius of the capsule in meters. |
| **Dimensions (Box Only)** | The x, y, and z dimensions of the box in meters. |

**Maximum Slope Angle**

The maximum slope angle is the largest slope that the character controller can climb. The character cannot move in directions that exceed this slope. If the character is standing on a slope above the maximum slope angle, its behavior depends on the slope behavior setting. The range of values allowed in the **Entity Inspector** is from 0 to 89 degrees.

**Step Height**

The maximum slope angle determines the step height that the controller can climb.

**Example**

Capsule controllers might be able to climb steps slightly higher than the step height because the curved bottom can slide upwards on steps. Refer to the following image:
The contact offset is the distance padding between the collider shape and the contact surface. The contact offset allows the simulation to provide smoother collision behavior.

**Note**

The contact offset is included in the calculation for the foot position.

**Example**

In the editor debug draw for the PhysX Character Controller component, the effect of the contact offset is represented by the wireframe that surrounds the solid shape of the collider, as in the following image.
Shape

You can use character controller collider with the following shapes:

- Capsule
- Box

Use the **Shape** property in the **Entity Inspector** to choose the desired shape. When you do so, the relevant dimensions are displayed for editing. The dimension settings are identical to the capsule and box options for the **PhysX Collider (p. 735)** component.

Differences Between PhysX and Legacy Character Physics Components

Character controllers are usually **kinematic** or **dynamic**. Dynamic character controllers are controlled through their velocity or by applying forces. Kinematic character controllers are controlled directly by position. Each controller type has advantages and disadvantages.
For more information, see Character Controllers in the NVIDIA documentation.

In Lumberyard, the PhysX Character Controller component is kinematic. The legacy Character Physics component is dynamic.

Because the PhysX Character Controller component is kinematic and not affected by outside forces, it is not affected by gravity out of the box. This separation allows you to use Script Canvas or C++ to implement custom behavior for gravity. Kinematic controllers behave as if they have infinite mass when dynamic objects collide with them. Your custom gameplay logic determines how the controller responds to collisions such as the recoil from heavy impacts.

**PhysX Collider**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Performance demands in games and real-time applications require physics simulations to be solved in fractions of a second. The PhysX Collider component allows you to specify primitive shapes or PhysX mesh assets to calculate collisions between entities, ensuring fast physics simulation. A simple entity such as a crate might have a single PhysX Collider component, while more complex entities, such as vehicles, might require multiple PhysX Collider components.

**Note**

The PhysX Collider component attached to an entity by itself creates a static (non-moving) entity, such as a wall or a mountain. To create a dynamic (moving) entity, you also need to add a PhysX Rigid Body component. The PhysX Rigid Body Physics component requires either a primitive collider or convex mesh collider. Triangle mesh physics assets work only with static entities.

The PhysX Collider component requires the PhysX gem enabled in your project.

For more information, see PhysX system.

**Topics**

- PhysX Collider properties
- Static PhysX entities
- Primitive colliders
- Physics asset colliders
- Collider component mode
- Colliders as triggers
PhysX Collider properties

Collision Layer
The collision layer that's assigned to the collider. For more information, see Collision Layers (p. 2785).

Collides With
The collision group containing the layers that this collider collides with. For more information, see Collision Groups (p. 2787).

Trigger
Set this collider as a trigger. A trigger performs a quick overlap test and does not apply forces or return contact point information. Use this to speed-up PhysX computations where a simple overlap between colliders is sufficient.

**Note**
Triangle meshes are not supported as triggers.
Trigger Area components cannot be used with PhysX Collider.

### Simulated
When enabled, this shape collider will be part of the physics simulation.

### In Scene Queries
When enabled, this shape collider can be queried for raycasts, shapecasts and overlap.

### Offset
Local offset position of the collider, relative to the entity.

### Rotation
Local rotation of the collider about the Offset of the PhysX collider component.

### Physics Material - Library
Set the physics material library for this collider.

### Physics Material - Mesh Surfaces
Choose a material from the physics material library for this collider. A collider can have multiple materials assigned. For more information, see Physics materials (p. 2797).

### Tag
Set a tag for this collider. Tags can be used to quickly identify components in script or code.

### Rest offset
PhysX bodies come to rest separated by the sum of their rest offset values. The Rest offset value must be less than the Contact offset value. Valid values range from \(-\infty\) to 50.

### Contact offset
PhysX bodies generate contacts when they are within the sum of their contact offset values. The Contact offset value must be greater than the Rest offset value. Valid values range from 0 to 50.

### Shape
Shape of the collider. A collider can be a primitive shape or a physics asset.

To use a primitive shape, choose Box, Sphere, or Capsule. For more information, see Primitive colliders (p. 739).

To use a physics asset, choose Physics Asset. For more information, see Physics asset colliders (p. 739).

### PhysX Mesh (Physics Asset shape)
Assign a physics asset to the collider. For more information, see FBX Settings PhysX export (p. 439).

### Radius (Sphere shape)
Radius multiplier of the sphere collider. The size of the sphere primitive is the Radius multiplied by the largest value in the Scale property in the Transform (p. 878) component.
Dimensions (Box shape)

Width, depth, and height of the box collider.

Height (Capsule shape)

Height of the capsule primitive shape. The height property of the capsule must be at least twice the radius property. For example, if the radius of the capsule is 5.0, the minimum height is 10.0.

Radius (Capsule shape)

Radius of the capsule primitive shape. The radius property of the capsule must be no greater than half the height property. For example, if the height of the capsule is 10.0, the maximum radius is 5.0.

Asset Scale

Scale the collider shape independent of the entity.

Physics Materials from Mesh

When the Physics Asset shape is selected, and Physics Materials from Mesh is enabled, the physics material for this collider is automatically set based on the surface type in the mesh's material. If the surface type in the mesh's material doesn't exist in the Physics Material - Library, the Physics Material - Mesh Surface is set to default. The collider Mesh Surfaces properties cannot be edited while this option is enabled.

Draw Collider

Render the collider in the viewport. Enabled by default.

Edit

Enable collider component mode to edit properties of the collider in the viewport using manipulators.

Static PhysX entities

A PhysX entity that is static can interact with other entities, but doesn't move.

To create a static PhysX entity

1. Create an entity. For more information, see Creating an Entity (p. 463).
2. In the Entity Inspector, choose Add Component and then select Mesh (p. 684) component.
3. In the Mesh component, choose a mesh asset for the Mesh asset property.
4. In the Entity Inspector, choose Add Component and then select PhysX Collider component.
5. In the **PhysX Collider** component, set the **Shape** to **Box**, and edit the **Dimensions** property so the box encloses the mesh asset.

6. Press **Control+G** to enter play mode. Because your entity does not have a **PhysX Rigid Body** component, it is static and does not move.

**Tip**
On the **Transform (p. 878)** component, enable the **Static** property. This enables optimizations for static entities.

### Primitive colliders

When you add the **PhysX Collider** component to an entity, you can specify the following basic collider shapes.

- **Sphere**
- **Box**
- **Capsule**

These primitive shapes don't have an underlying mesh. Because they are defined by dimensions rather than a mesh, they are high-performance colliders and should be used when possible.

### Physics asset colliders

Physics asset colliders are meshes that are created in a modeling application, or are convex meshes that are automatically generated by the FBX exporter. Because physics asset colliders are more complex than shapes, they are less efficient. Physics asset colliders should be used in cases where collision detection that more closely resembles the complex shape of the visible mesh is required. To generate PhysX collider mesh assets for your project, see **FBX Settings PhysX export (p. 439)**.
To define a mesh collider that has varying properties:

- Use a third-party content creation tool to define a mesh collider and use the FBX exporter to convert the mesh collider for your project. Mesh colliders created this way can only be added to static entities.
- Alternatively, attach multiple PhysX Collider components to the entity, and specify different PhysX collider shapes and properties for each component.

To create a mesh collider

1. Create an entity. For more information, see Creating an Entity (p. 463).
2. In the Entity Inspector, choose Add Component and then select Mesh (p. 684).
3. In the Mesh component, choose a mesh asset for the Mesh asset property.
4. In the Entity Inspector, choose Add Component and then select PhysX Collider.
   
   **Note**
   
   If the asset specified for the Mesh component contains a PhysX collider mesh asset, the PhysX Collider automatically sets its Shape property to Physics Asset, and its PhysX Mesh property to the PhysX collider mesh asset. If the asset specified for the Mesh component contains more than one PhysX collider mesh asset, the first PhysX collider mesh asset found is assigned to the PhysX Mesh property.

5. In the PhysX Collider properties, ensure that the Shape property has Physics Asset selected.
6. For PhysX Mesh, ensure that the desired PhysX collider mesh asset is selected. Click the ... button to the right of PhysX Mesh to change the mesh asset.

**Note**

To generate PhysX collider mesh assets for your project, see FBX Settings PhysX export (p. 439).

**Example**

Instead of a primitive shape, the entity has a PhysX collider mesh asset specified for the PhysX Collider component.
Note
To make an entity dynamic, in the Entity Inspector, choose Add Component and then select PhysX Rigid Body Physics component. Only primitive shapes and convex meshes can be used for dynamic colliders. If you assign a triangle mesh, the collider won't work. For dynamic objects, be sure to disable the Static property of the Transform component of your entity.

Collider component mode

In collider component mode, you edit colliders with manipulators in the viewport. To enter collider component mode, choose the Edit button at the bottom of the PhysX Collider component properties in the Entity Inspector.

Sub component modes

There are three editing modes available in collider component mode.

- **Resize** mode, which is unique to each collider type, scales the collider.

  The manipulator displayed in the viewport in resize mode is dependent on the collider shape. For primitive colliders, the resize manipulator handles are represented as black squares. For Physics Asset colliders, the resize manipulator is represented as a familiar scale manipulator.

- **Offset** mode translates the collider relative to its entity transform.

- **Rotation** mode rotates the collider about the component's Offset.
Resize (Sphere Shape)

*Sphere* resize mode has one linear manipulator that controls the *Radius* property.

Resize (Box Shape)

*Box* resize mode has six linear manipulators, one on each side of the box. The manipulators control the width, depth, and height *Dimensions* property.
**Resize (Capsule Shape)**

Capsule resize mode has two linear manipulators. The manipulator at the top of the capsule controls the **Height** property. The manipulator on the side controls the **Radius** property.

**Resize (Physics Asset Shape)**

Physics Asset resize mode has a three axis scale manipulator.

**Offset**

Offset mode has a three axis translate manipulator.
Rotation

Rotation mode has a three axis rotate manipulator.

Collider component mode hotkeys

These navigation hotkeys are available in collider component mode.

• 1 – Resize mode.
• 2 – Offset mode.
• 3 – Rotation mode.
• Control + Mouse Wheel Up – Next mode.
• Control + Mouse Wheel Down – Previous mode.
• R – Reset current mode. This is effectively an undo operation. You can step through the Resize, Offset, and Rotation modes and press R to reset changes to the current mode.
• Escape – Exit component mode.

Colliders as triggers

Triggers allow colliders to perform efficient overlap tests. Colliders marked as triggers won't have forces applied when they intersect with another collider. This is useful for detecting when something enters a certain area or when two objects overlap. Use Lua or Script Canvas to detect overlap.

Note
Because triggers don't perform contact resolution, the contact points between a trigger and another collider aren't available.
Triangle meshes are not supported as triggers.
Trigger Area components cannot be used with PhysX Collider.

PhysX Fixed Joint component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With the PhysX Fixed Joint component, you can create a dynamic fixed joint that constrains an entity to the joint with no degree of freedom in any axis.

PhysX Fixed Joint component properties
Local Position

Specify the position of the joint relative to the entity transform.

Local Rotation

Specify the rotation of the joint relative to the entity transform.

Lead Entity

Specify the parent entity that will drive the joint.

Breakable

When enabled, the joint will break if sufficient force is applied. Enabling Breakable exposes the Maximum Force and Maximum Torque properties.

Note

PhysX Rigid Body components that have their Compute Mass property enabled might have very large mass values. If the entity containing the joint component or the leader entity have their Compute Mass property enabled, the Maximum Force and Maximum Torque properties might require very high values to resist breaking.

Maximum Force

When Breakable is enabled, specify the maximum force the joint can sustain before breaking. Valid values range from 0.01 to Infinity.

Maximum Torque

When Breakable is enabled, specify the maximum torque the joint can sustain before breaking. Valid values range from 0.01 to Infinity.

Display Setup in Viewport

When enabled, a line that represents the connection between the joint and its leader are displayed.

Select Lead on Snap

When enabled, snapping the joint to an entity in component mode will set the entity as the Lead Entity. The entity containing the joint component is excluded from this operation.

Lead-Follower Collide

When enabled, the lead entity and follower entity (the entity containing the joint component) will collide.

Edit

When clicked, component edit mode is enabled. In component edit mode, all components are locked except for the PhysX Ball Joint component. The properties of the PhysX Ball Joint component can be edited in Perspective. Press Tab to cycle through the component edit modes. Click Done to exit component mode.

PhysX Force Region

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the PhysX Force Region component to specify a region that applies physical force to entities. For each physics simulation frame, the component applies force to entities that are in the
region's bounds. You can use this component to simulate effects such as simulating gravity, slowing down, or deflecting an entity to another direction.

To create a force region, you must do the following:

- Enable the PhysX (p. 1188) gem for your game project
- Add a PhysX Collider (p. 735) component to the same entity
- For the PhysX Collider component, you must select the Trigger property for the force region to work

The PhysX Collider component's shape, size, and orientation represent the region that applies force to incoming entities.

**Note**

- For the Shape property, if you select PhysicsAsset, you must select a PxMesh asset file. If the asset isn't a convex mesh (for example, it's a triangle mesh), the collider won't work as a trigger for collisions.

**Topics**

- PhysX Force Region Component Properties (p. 748)
- Creating a Force Region (p. 752)
PhysX Force Region Component Properties

Visible
Debug Forces

Forces
6 elements

[0]
Force Type
World Space

World Space Force
Direction
X 0.0
Y 0.0
Z 1.0
Magnitude
10.0

[1]
Force Type
Local Space

Local Space Force
Direction
X 0.0
Y 0.0
Z 1.0
Magnitude
10.0

[2]
Force Type
Point

Point Force
Magnitude
1.0

[3]
Force Type
Spline Follow

Spline Follow Force
Damping Ratio
1.0
Frequency
3.0
Target Speed
1.0
Lookahead
0.0

[4]
Force Type
Simple Drag

Simple Drag Force
Region Density
1.0

[5]
Force Type
Linear Damping

Linear Damping Force
Damping
1.0
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
<tr>
<td>Debug Forces</td>
<td>A debug arrow draws in gameplay mode. The debug arrow indicates the direction of the net force for each entity inside the force region.</td>
</tr>
<tr>
<td>Forces</td>
<td>Specifies the force types that act in the force region. You can add multiple force types for the same component.</td>
</tr>
</tbody>
</table>

**Force Types**

You can add multiple force types to the component. When an entity enters the force region, the entity moves according to the net value of the forces that you specify.

**Contents**
- Linear Damping (p. 749)
- Local Space (p. 749)
- Point (p. 750)
- Simple Drag (p. 750)
- Spline Follow (p. 750)
- World Space (p. 751)

**Linear Damping**

Applies a force in the opposite direction to an entity's velocity. For example, you can create a force that simulates a swamp or mud.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Damping</td>
<td>The amount of damping to apply. You can't specify negative values. Specify higher values to apply more damping. Specify 0 for no damping.</td>
</tr>
</tbody>
</table>

**Local Space**

Applies a force in local space, relative to the force region's orientation. For example, you can create a force that simulates a hair dryer or a vacuum cleaner.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>The direction of the force in the local space of the force region.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>You can specify a value from -1000000 to 1000000, but Lumberyard Editor normalizes the value to a range of -1 and 1.</td>
</tr>
</tbody>
</table>

**Magnitude**

The amount of force to apply. Specify a negative value to apply the force in the opposite direction.

**Point**

Applies a force relative to the center of force region. The magnitude determines if the force is inward or outward. For example, you can create a force that simulates an explosion or a black hole.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Magnitude</strong></td>
<td>The amount of force to apply. Specify a positive value for an outward force and a negative value for an inward force.</td>
</tr>
</tbody>
</table>

**Simple Drag**

Applies a force that simulates air resistance. **Simple Drag** always applies force in the opposite direction of colliding entities. Larger and faster entities experience more drag. Entities are approximated as spheres.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Region Density** | The density of the volume. Specify higher values to increase the drag force.  
**Note** | You can specify only positive values. |

**Spline Follow**

Applies a force to make entities follow a spline. The force uses a proportional-derivative (PD) controller that simulates a spring moving along a spline. For example, you can create a force that simulates a water slide.

**Note**

- For the force region entity, if you change the **Scale** property of the **Transform** component, the scaling must be uniform so that the x, y, and z scale values match. If scaling isn’t uniform, the spline doesn’t correctly reflect the path of the force.
- The end of the spline must be outside the force region so that entities can exit after following the spline.
### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Damping Ratio</strong></td>
<td>Slows down the vibration of the entity when it enters the spline.</td>
</tr>
<tr>
<td></td>
<td>The higher the value, the faster it slows down the vibrations.</td>
</tr>
<tr>
<td></td>
<td>A value of 1 quickly slows down vibrations around the spline.</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>The frequency of the vibration when the entity enters the spline.</td>
</tr>
<tr>
<td><strong>Target Speed</strong></td>
<td>The speed that the entity attempts to reach as it travels through the nodes of the spline.</td>
</tr>
<tr>
<td></td>
<td>Specify a negative value to apply the force in the opposite direction.</td>
</tr>
<tr>
<td><strong>Lookahead</strong></td>
<td>The distance, in meters, that entities look ahead in their path to reach a node on the spline.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>The Lookahead value determines how far ahead entities look for the next node to steer towards. At each physics simulation frame, the entity in the spline changes its direction to realign itself to the identified node.</td>
</tr>
<tr>
<td></td>
<td>If spline nodes are close together, you can specify a smaller Lookahead value so that entities can detect the next node.</td>
</tr>
<tr>
<td></td>
<td>If nodes are far apart, you can specify a higher Lookahead value so that entities can detect the next node.</td>
</tr>
</tbody>
</table>

### World Space

Applies a force in world space. World space force doesn't take into account an entity's orientation. For example, you can create a force that simulates gravity.

**Note**

You can define the direction for world space so that it always applies force in the direction that you want, regardless of the colliding entity.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Direction</strong></td>
<td>The direction of the force in world space.</td>
</tr>
<tr>
<td><strong>Magnitude</strong></td>
<td>The amount of force to apply. Specify a negative value to apply the force in the opposite direction.</td>
</tr>
</tbody>
</table>

**Note**

When you select a force type, remember the following:
• For **Simple Drag**, you can't define the direction of force. **Simple Drag** always works in the opposite direction of the entity's movement. In contrast, you can define a direction of force using **World Space**, which always acts in the direction that you specify, regardless of the direction of the moving entity.

• To determine how much force to apply, **Linear Damping** takes into account the colliding entity's velocity and mass but not its shape. In contrast, **Simple Drag** takes into account the colliding entity's velocity, cross-section area, and the **Region Density** of the force region.

### Creating a Force Region

You can create a force region so that force applies to another entity that enters the region.

**To create a PhysX Force Region component**

1. In Lumberyard Editor, create an entity. For more information, see Creating an Entity (p. 463).
2. Enter a name for the entity, such as **ForceRegion**.
3. In the **Entity Inspector**, choose **Add Component** and select a **PhysX Force Region** component.
4. Choose **Add Required Component** and select the **PhysX Collider** component.
5. For the **PhysX Collider** component, do the following.
   a. Select the **Trigger** property.
   b. For **Shape**, select a shape such as **Box**.
6. For the **PhysX Force Region** component, do the following.
   a. Select the **Visible** and **Debug Forces** properties.
   b. For **Forces**, click the + icon and for **Force Type**, select a force such as **Local Space**.
   c. For **Magnitude**, enter a value such as **20**.

   Blue arrows appear on the entity that indicate the direction of the force.

7. To collide an entity with the force region, create a dynamic entity named **Sphere** and attach the **PhysX Collider** and **PhysX Rigid Body Physics** components. These components enable the entity to interact with other PhysX entities.
8. (Optional) Add a **Mesh** component and, for **Mesh asset**, select a mesh asset, such as a **primitive_sphere.cgf**.
9. Select and drag the **Sphere** entity so that it's above the force region.
10. After you create your dynamic entity, press **Ctrl+G** to enter gameplay mode.

**Example**

The sphere falls and collides with the force region. The force region applies force and pushes the sphere in the opposite direction.
11. To leave gameplay mode, press Esc.

**Note**
To display PhysX debug visualizations, see Debugging PhysX (p. 2826).
For more information about using PhysX components, see *PhysX system* (p. 2772).
PhysX Hinge Joint component

With the PhysX Hinge Joint component, you can create a dynamic hinge joint that constrains an entity to the joint with freedom to rotate around the x-axis of the joint.

PhysX Hinge Joint component properties

Local Position

Specify the position of the joint relative to the entity transform.

Local Rotation

Specify the rotation of the joint relative to the entity transform.

Lead Entity

Specify the parent entity that will drive the joint.
Breakable

When enabled, the joint will break if sufficient force is applied. Enabling Breakable exposes the Maximum Force and Maximum Torque properties.

Note
PhysX Rigid Body components that have their Compute Mass property enabled might have very large mass values. If the entity containing the joint component or the leader entity have their Compute Mass property enabled, the Maximum Force and Maximum Torque properties might require very high values to resist breaking.

Maximum Force

When Breakable is enabled, specify the maximum force the joint can sustain before breaking. Valid values range from 0.01 to Infinity.

Maximum Torque

When Breakable is enabled, specify the maximum torque the joint can sustain before breaking. Valid values range from 0.01 to Infinity.

Display Setup in Viewport

When enabled, three planes are displayed showing the orientation and limits of the joint. Red and green planes show the Positive angular limit and Negative angular limit. The white plane shows the 0 degree rotation of the joint. The shared edge of the three planes is the x-axis of the hinge joint. A line displays the connection between the joint and the Lead Entity.

Select Lead on Snap

When enabled, snapping the joint to an entity in component mode will set the entity as the Lead Entity. The entity containing the joint component is excluded from this operation.

Lead-Follower Collide

When enabled, the lead entity and follower entity (the entity containing the joint component) will collide.

Limit

When enabled, the lead entity's movement around the joint axis is constrained by angular limits. Enabling Limit exposes the Soft Limit, the Positive angular limit, and the Negative angular limit properties.

Soft Limit

When enabled, the lead entity's movement around the joint axis is allowed to pass the specified limit. With Soft Limit enabled, when the lead entity rotates past the limit, the lead entity's movement is treated as a spring and will slow then spring back to the limit area. Enabling Soft Limit exposes the Damping and Stiffness properties.

Damping

When Soft Limit is enabled, the spring's drive relative to the velocity of the follower when outside the rotation limit. Valid values range from 0.001 to 1000000.0.

Stiffness

When Soft Limit is enabled, the spring's drive relative to the position of the follower when outside the rotation limit. Valid values range from 0.001 to 1000000.0.

Positive angular limit

When Limit is enabled, the positive rotation limit around the joint's axis. Valid values range from 0.1 to 360.0.

Negative angular limit

When Limit is enabled, the negative rotation limit around the joint's axis. Valid values range from 0.1 to 360.0.
**Edit**

When clicked, component edit mode is enabled. In component edit mode, all components are locked except for the **PhysX Ball Joint** component. The properties of the **PhysX Ball Joint** component can be edited in **Perspective**. Press **Tab** to cycle through the component edit modes. Click **Done** to exit component mode.

---

**PhysX Ragdoll**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the AWS Game Tech blog to learn more.

You can use the **PhysX Ragdoll** component to create a physical representation of a character in the animation system and to simulate certain behaviors, such as hit reactions and character death. The physical representation consists of a hierarchy of rigid bodies with simple shapes that are connected by joints. You can adjust the ragdoll settings as needed for physical plausibility and performance.

The **PhysX Ragdoll** component requires the **PhysX Characters** (p. 1188) gem.

For more information about the PhysX system, see **Simulating physics behavior with the PhysX system** (p. 2772).

---

**Using the PhysX Ragdoll Component**

You use the PhysX system and the **Animation Editor** to create a ragdoll.

**To use the PhysX Ragdoll component**

1. In Lumberyard Editor, add the **PhysX Ragdoll** component to an entity that represents a character. For more information, see **Adding Components to an Entity** (p. 479).
2. Choose **Tools**, **Animation Editor**.
3. Use the **Animation Editor** to create and control the physical representation of the ragdoll. For more information, see **Creating and Simulating a PhysX Ragdoll** (p. 1458).

---

**PhysX Ragdoll Component Properties**

![PhysX Ragdoll Component Properties](image)
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Position Iteration Count</strong></td>
<td>Specifies the number of iterations to use for joint stability and accuracy. A higher iteration count produces more realistic behavior but decreases performance. A lower iteration count may result in unrealistic behavior, such as joints separating and parts of the ragdoll intersecting with the terrain. Default: 16 Valid values: 1 to 255</td>
</tr>
<tr>
<td><strong>Velocity Iteration Count</strong></td>
<td>Specifies the number of iterations to use for resolving collisions, such as restitution (bounciness) and rigid body intersection. A higher iteration count resolves collisions based on the material settings but decreases performance. Use a lower iteration count to reduce the restitution for your ragdoll. Default: 8 Valid values: 1 to 255</td>
</tr>
<tr>
<td><strong>Enable Joint Projection</strong></td>
<td>If set, joint projection preserves joint constraints in demanding situations, such as parts of the ragdoll moving energetically. This setting may improve physical plausibility. Enabled by default.</td>
</tr>
<tr>
<td><strong>Joint Projection Linear Tolerance</strong></td>
<td>The maximum linear deviation that the PhysX system allows in each joint. Values less than 0.001 meters are not recommended due to jittering. To edit this property, you must set the <strong>Enable Joint Projection</strong> property. Default: 0.001</td>
</tr>
<tr>
<td><strong>Joint Projection Angular Tolerance</strong></td>
<td>Maximum angular deviation that the PhysX system allows in each joint. Values less than 1 degree are not recommended due to jittering. To edit this property, you must set the <strong>Enable Joint Projection</strong> property. Default: 1</td>
</tr>
</tbody>
</table>
PhysX Rigid Body

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The PhysX Rigid Body component defines the entity as a rigid object that is solid and can move and collide with other PhysX entities. For example, you can add the PhysX Rigid Body component to the entity to create a projectile.

You can specify two main modes for a PhysX Rigid Body component.

Dynamic

Dynamic rigid bodies will be fully simulated by Lumberyard and respond to collision events with other rigid bodies. Lumberyard will apply forces to two dynamic objects that collide, which results in a realistic physics simulation. The simulation will also apply a gravity force to rigid bodies in dynamic mode. You can disable this feature for each rigid body on the component. Dynamic rigid bodies should be used for semi-realistic solid objects that are effected by gravity and other forces.

Dynamic is the default.

Kinematic

Kinematic rigid bodies are not fully simulated. Kinematic rigid bodies are moved by script and not effected by forces or gravity. Movement is created with the SetKinematicTarget method that you specify in the script.

Use this feature for objects such as moving platforms, doors, or anything that doesn't need to be fully simulated by physics.

Note

You should always add the PhysX Rigid Body component to the top level of an entity hierarchy. If you add the component to a child entity, this can cause conflicts with the entity's world transform and result in undefined behavior.

The PhysX Rigid Body components requires the PhysX (p. 1188) gem.

For more information, see PhysX system (p. 2772).

Topics

- PhysX Rigid Body component properties (p. 760)
- Creating a Dynamic PhysX Entity (p. 762)
PhysX Rigid Body component properties

The **PhysX Rigid Body** component has the following properties.

**Initial linear velocity**

Specifies the starting linear velocity of the rigid body when spawned. This creates movement in the direction of the linear velocity.

**Initial angular velocity**

Specifies the starting angular velocity of the rigid body when spawned. This creates rotation in the direction of the angular velocity.

**Linear damping**

Specifies the rate of decay over time for linear velocity even if no forces are acting on the rigid body. A non-zero eventually stops the rigid body if no linear force is applied.
Value must be non-negative between 0 and \textit{infinity}.

**Angular damping**

Specifies the rate of decay over time for angular velocity even if no forces are acting on the rigid body. A non-zero eventually stops the rigid body if no torque force is applied.

Value must be non-negative between 0 and \textit{infinity}.

**Sleep threshold**

Specifies the kinetic energy per unit mass below which the rigid body can go to sleep.

Value must be non-negative between 0 and \textit{infinity}.

**Start asleep**

When enabled, the rigid body component is asleep when the entity is spawned, and will wake when a sufficient force is applied.

**Interpolate motion**

When enabled, the resulting motion from the simulation is smoothed.

Enable this property for objects that require smooth motion such as vehicles.

**Gravity enabled**

When enabled, the rigid body is effected by gravity. This only applies to dynamic rigid bodies.

**Kinematic**

When enabled, the rigid body is kinematic. The rigid body is not be effected by gravity or other forces and is moved by script.

When disabled, the rigid body is dynamic. The rigid body responds to gravity and other forces and its motion is simulated by PhysX.

**CCD enabled**

When enabled, the rigid body is will have continuous collision detection (CCD). This property is useful for high speed objects to ensure accurate collision detection. Enabling continuous collision detection reveals two additional properties, \textit{Min advance coefficient} and \textit{CCD Friction}.

**Note**

To set this property, you must also set the \textit{Continuous Collision Detection} in the \textit{PhysX Configuration} window. See World Configuration (p. 2775).

**Min advance coefficient**

Lower values reduce clipping but can affect motion smoothness.

The value must be a decimal between 0.01 and 0.99.

**CCD friction**

When enabled, friction is applied when CCD collisions are resolved.

**Compute mass**

When enabled, mass is computed for the rigid body.

**Mass**

When \textit{Compute Mass} is disabled, a \textit{Mass} value can be specified for the PhysX rigid body. Valid \textit{Mass} values range for 0 to \textit{infinity}. 
**Compute COM**

When enabled, the center of mass is computed for the rigid body.

**COM offset**

When **Compute COM** is disabled, the center of mass can be specified as an offset.

**Compute inertia**

When enabled, inertia is computed based on the mass and shape of the rigid body.

**Inertia diagonal**

When **Compute inertia** is disabled, an **Inertia diagonal** can be specified as the diagonal elements of the inertia tensor.

**Maximum angular velocity**

Angular velocity is clamped to the specified value. Valid values range from 0 to *infinity*.

**Include non-simulated shapes in Mass**

When enabled, non-simulated shapes will be included in the mass, center of mass, and inertia calculations.

**Debug draw COM**

When enabled, the center of mass is displayed for this PhysX rigid body.

---

**Creating a Dynamic PhysX Entity**

A PhysX entity that is dynamic can move and collide with other entities.

**To create a dynamic PhysX entity**

1. Create an entity. For more information, see Creating an Entity (p. 463).
2. In the **Entity Inspector**, choose **Add Component** and then select a **Mesh** (p. 684) component.
3. For **Mesh asset**, select the mesh asset so that your entity is visible, such as a box .cgf.
4. Add the **PhysX Collider** component to the entity.
5. Add the **PhysX Rigid Body** component to the entity.
6. Create another entity for your PhysX terrain. For more information, see the **PhysX Terrain** (p. 767) component.
7. Press **Ctrl+G** to enter gameplay mode.

**Example**

The entity has a **PhysX Collider** component and a **PhysX Rigid Body** component attached. Because the object is dynamic, it falls and then collides with the **PhysX Terrain** component.
Note

- For the Transform (p. 878) component, clear the Static property. This ensures that the mesh moves with the physics simulation.

PhysX Shape Collider

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The PhysX Shape Collider component creates NVIDIA PhysX simulation collider geometry based on the shape definition supplied by a Shape component. The PhysX Shape Collider supports the following Shape components:

- Box (p. 820)
• Capsule (p. 821)
• Polygon Prism (p. 708)
• Sphere (p. 824)

Note

The PhysX Shape Collider component attached to an entity with a supported Shape component creates a static (non-moving) entity. To create a dynamic (moving) entity, you also must add a PhysX Rigid Body (p. 759) component.

Although the PhysX Shape Collider is similar to the PhysX Collider (p. 735) component, you might want to use the PhysX Shape Collider instead in these scenarios:

• The shape information defined by the Shape component is used elsewhere in code or script. For example, the shape defines another volume, such as an audio volume or fog volume, and you want to keep the collider geometry and volume synchronized.
• You want to use a Shape component such as Polygon Prism Shape (p. 708) that is not provided by PhysX Collider.
• You have existing Shape components and don't want to migrate them to use PhysX Collider components.

The PhysX Shape Collider component has some limitations compared to the PhysX Collider component:

• Only one Shape component can be used per entity, and so only one PhysX Shape Collider component is supported per entity. Any number of PhysX Collider components can also be used on the same entity, however.
• The position and rotation of the PhysX Shape Collider component can't be offset relative to the entity position.

To use the PhysX Shape Collider component you must enable PhysX (p. 1188) gem in your project.

For more information, see PhysX system (p. 2772).

Topics

• PhysX Shape Collider properties (p. 765)
• Complex polygon prism shapes (p. 766)
• Colliders as triggers (p. 745)
PhysX Shape Collider properties

**Collision Layer**

The collision layer that's assigned to this shape collider. For more information, see Collision Layers (p. 2785).

**Collides With**

The collision group containing the layers that this shape collider collides with. For more information, see Collision Groups (p. 2787).

**Trigger**

When enabled, this shape collider functions as a trigger. A trigger performs a quick overlap test and does not apply forces or return contact point information. Use this to speed up PhysX computations where a simple overlap between colliders is sufficient.

**Note**

Trigger Area components are legacy components and cannot be used with PhysX Shape Collider.

**Simulated**

When enabled, this shape collider will be part of the physics simulation.

**In Scene Queries**

When enabled, this shape collider can be queried for raycasts, shapecasts and overlap.

**Physics Material - Library**

Set the physics material library for this shape collider.

**Physics Material - Mesh Surfaces**

Choose a material from the physics material library for this shape collider.
Tag

Set a tag for this shape collider. Tags can be used to quickly identify components in script or code.

Rest offset

PhysX bodies come to rest separated by the sum of their rest offset values. The Rest offset value must be less than the Contact offset value. Valid values rage from $-\infty$ to 50.

Contact offset

PhysX bodies generate contacts when they are within the sum of their contact offset values. The Contact offset value must be greater than the Rest offset value. Valid values rage from 0 to 50.

Draw collider

Render this shape collider in the viewport. Enabled by default.

Complex polygon prism shapes

The Polygon Prism Shape (p. 708) is automatically subdivided into convex portions, which means that polygon prisms can be used with dynamic rigid bodies or as triggers in PhysX. The subdivision is automatically updated if the vertices of the polygon prism are modified.

If the vertices are modified so that the polygon prism is no longer a simple polygon, for example, if the polygon prism is self-intersecting, it isn't possible to subdivide the polygon prism into convex pieces. An error will display in the Editor Console, as shown in the following example.
**Colliders as triggers**

Triggers allow colliders to perform efficient overlap tests. Colliders marked as triggers won't have forces applied when they intersect with another collider. This is useful for detecting when something enters a certain area or when two objects overlap. Use Lua or Script Canvas to detect overlap.

**Note**

Because triggers don't perform contact resolution, the contact points between a trigger and another collider aren't available. Trigger Area components are legacy components and cannot be used with PhysX Shape Collider.

**PhysX Terrain**

**Important**

The PhysX Terrain component and associated documentation are legacy features retained for compatibility. For new projects, use the **Legacy Terrain** level component, and the **PhysX Terrain** level component. For more information, see [Legacy Terrain level component (p. 651)](#).

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the [AWS Game Tech blog](https://aws.amazon.com/gaming/) to learn more.

The **PhysX Terrain** component exports and saves the terrain as an asset that loads at runtime. The **PhysX Terrain** component is required for games in which physics interacts with the terrain. For example, you can create a terrain collider so that your entities can interact with it, such as a barrel that falls to the ground and then rolls to a stop.

For more information about using PhysX components, see [PhysX system (p. 2772)](#).

**Topics**

- [PhysX Terrain Component Properties (p. 748)](#)
- [Creating a PhysX Terrain Collider (p. 768)](#)

### PhysX Terrain Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision Layer</td>
<td>Specifies on which <strong>Collision Layer</strong> the terrain exists.</td>
</tr>
<tr>
<td></td>
<td>You can configure <strong>Collision Layer</strong> settings in the <strong>PhysX Configuration</strong> window.</td>
</tr>
</tbody>
</table>
Creating a PhysX Terrain Collider

As a best practice, create a top-level entity in your level and add the PhysX Terrain component. You only need one entity with the PhysX Terrain attached for your level.

To create a PhysX Terrain Collider

1. In Lumberyard Editor, create an entity. For more information, see Creating an Entity (p. 463).
2. Enter a name for the entity, such as terrain.
3. In the Entity Inspector, choose Add Component and then select a PhysX Terrain component. The terrain collider appears in the viewport.

   **Note**
   The Transform component for the entity is ignored and doesn't affect the position of the terrain.

4. To have an entity interact with the terrain collider, you can create a dynamic entity so that the two entities interact. For more information, see Creating a Dynamic PhysX Entity (p. 762).
5. After you create your dynamic entity, press Ctrl+G to enter gameplay mode. The dynamic entity falls and then collides with the terrain collider.
6. Press Esc to leave gameplay mode.

   **Note**
   In some cases, performance can be impacted when a large body intersects the terrain. To overcome this issue, you can clear the Persistent Contact Manifold option in the World Configuration (p. 2775). If you clear this option, keep the size of the colliders that intersect the terrain small.

Rain

This feature is in preview release and is subject to change.
You can use the **Rain** component to create rain effects, puddles, and ripples on the terrain. You can add multiple **Rain** components to your entities in a level, but the level uses only one of the **Rain** components. The last **Rain** component to activate takes priority and Lumberyard renders that component.

To enable the **Rain** component, you must enable the Rain gem. For more information, see [Rain Gem](p. 1194).

**Note**
If your level has a [Snow](p. 853) and **Rain** component, the component that activates last takes priority; you cannot enable both **Snow** and **Rain** components at the same time.

**Contents**
- Turning Rain On or Off (p. 769)
- Editing Rain Effects (p. 770)
- Rain Component Properties (p. 770)
- EBus Request Bus Interface (p. 772)
  - Request Bus Example Script (p. 776)

**Turning Rain On or Off**

You can toggle rain on or off in Lumberyard Editor by enabling or disabling **AI/Physics**, or by running or stopping the game.

**To toggle rain effects on or off in Lumberyard Editor**
- Do one of the following:
On the bottom right of the viewport, click **AI/Physics**.

Choose **Game, Enable Physics/AI**.

Press **Ctrl+P**.

Press **Ctrl+G** to enter gameplay mode. Press **Esc** to stop the game.

## Editing Rain Effects

You can edit rain effects only when the **AI/Physics** toggle is disabled.

**To edit rain effects**

1. Disable **AI/Physics**.
2. Make changes to the **Rain** component properties.
3. Enable **AI/Physics**. After you reenable **AI/Physics**, your changes appear in the viewport.

### Rain Component Properties

The **Rain** component has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabled</strong></td>
<td>Specifies whether rain effects are enabled.</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>True</strong></td>
</tr>
<tr>
<td><strong>Use VisArea</strong></td>
<td>Specifies whether rain appears when the player is inside of a visible area.</td>
</tr>
<tr>
<td></td>
<td>For more information, see the <strong>VisArea</strong> component.</td>
</tr>
</tbody>
</table>
## Property Description

### Disable Occlusion
Specifies whether objects will ignore rainfall effects. You can enable the Rain occluder parameter in the Mesh component for your objects; rain effects do not affect these objects.

For more information, see Mesh Component: Advanced Options (p. 685).

Default value: `false`

### Radius
Radius of the area where puddles are created. Rain continues to affect the screen even outside the specified radius.

Default value: 10000

Valid values: 0 to 10000

### Amount
Amount of rain and other effects that the Rain component creates.

Default value: 1

Valid values: 0 to 100

### Diffuse Darkening
Amount of darkening that is applied to surfaces that the rain affects.

Default value: 0.5

Valid values: 0 to 1

The Rain component has the following raindrop options.

### Amount
Quantity of raindrops.

Default: 0.5

Valid values: 0 to 100

### Speed
Speed of the falling raindrops.

Default: 1.0

Valid values: 0 to 100

### Lighting
Brightness of the raindrops.

Default: 1.0

Valid values: 0 to 100

The Rain component has the following puddle options.
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>Amount of depth and brightness of rain puddles.</td>
</tr>
<tr>
<td></td>
<td>Default: 1.5</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 10</td>
</tr>
<tr>
<td>Mask Amount</td>
<td>Strength of the puddle mask. A value of 1 creates puddles of rain between patches of land.</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1</td>
</tr>
<tr>
<td>Ripple Amount</td>
<td>Strength and frequency of the ripples in the rain puddles.</td>
</tr>
<tr>
<td></td>
<td>Default: 2</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Splashes Amount</td>
<td>Strength and frequency of the splashes that the raindrops create.</td>
</tr>
<tr>
<td></td>
<td>Default: 1.3</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1000</td>
</tr>
</tbody>
</table>

### EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game. You can use this EBus to communicate to an entity with a Rain component attached. The EBus is available at game run time and editing and can be accessed from C++, Lua, and the Script Canvas editor.

When set, the individual setters on this bus update the rain simulation immediately. This can cause performance implications if you want to change multiple parameters. To avoid this, you can set multiple parameters at once with GetRainOptions (p. 775) and SetRainOptions (p. 775).

For more information, see Working with the Event Bus (EBus) system (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Enables rain effects.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables rain effects.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Toggle</td>
<td>Toggles whether rain effects are enabled.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>IsEnabled</td>
<td>Returns true if rain effects are enabled.</td>
<td>None</td>
<td>Type: Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>SetIgnoreVisAreas</td>
<td>Specifies whether rain effects ignore visible areas. If true, the rain effects are visible, even inside visible areas. If false, rain does not render in a visible area. For more information, see the VisArea (p. 918) component.</td>
<td>Type: Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetIgnoreVisAreas</td>
<td>Returns true if rain effects ignore visible areas.</td>
<td>None</td>
<td>Type: Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>SetDisableOcclusion</td>
<td>Specifies whether rain effects affect objects that are marked as occluders. You can enable the Rain occluder parameter in the Mesh component for your objects; rain effects do not affect these objects. For more information, see Mesh Component: Advanced Options (p. 685). If true, objects marked as rain occluders are affected by rain effects. If false, objects marked as rain occluders ignore rain effects.</td>
<td>Type: Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetDisableOcclusion</td>
<td>Returns true if rain effects affect objects that are marked as occluders.</td>
<td>None</td>
<td>Type: Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>SetRadius</td>
<td>Sets the radius of rain effects. The Rain component creates puddles inside only the specified radius. Raindrops always render in the level.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetRadius</td>
<td>Returns the radius of rain effects.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetAmount</td>
<td>Sets the overall amount of rain, puddles, and splashes.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetAmount</td>
<td>Returns the overall amount of rain, puddles, and splashes.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetDiffuseDarkening</td>
<td>Sets the amount of darkening that is applied to surfaces that the rain affects.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetDiffuseDarkening</td>
<td>Returns the amount of darkening that is applied to surfaces that the rain affects.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetRainDropsAmount</td>
<td>Sets the amount of raindrops that appear.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetRainDropsAmount</td>
<td>Returns the quantity of raindrops that appear.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetRainDropsSpeed</td>
<td>Sets the speed of raindrops.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetRainDropsSpeed</td>
<td>Returns the speed of raindrops.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetRainDropsLight</td>
<td>Sets the brightness of the raindrops.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetRainDropsLight</td>
<td>Returns the brightness of the raindrops.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------</td>
<td>------------</td>
<td>----------------</td>
<td>------------</td>
</tr>
<tr>
<td>SetPuddlesAmount</td>
<td>Sets the amount of depth and brightness of the puddles.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetPuddlesAmount</td>
<td>Returns the amount of depth and brightness of the puddles.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetPuddlesMaskAmount</td>
<td>Sets the strength of the puddle mask applied to the puddles.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetPuddlesMaskAmount</td>
<td>Returns the strength of the puddle mask applied the puddles.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetPuddlesRippleAmount</td>
<td>Sets the strength and frequency of the ripples in puddles.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetPuddlesRippleAmount</td>
<td>Returns the strength and frequency of the ripples in puddles.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSplashesAmount</td>
<td>Sets the strength of the splash effects that the rain creates.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSplashesAmount</td>
<td>Returns the strength of the splash effects that the rain creates.</td>
<td>None</td>
<td>SnowOptions</td>
<td>Yes</td>
</tr>
<tr>
<td>SetRainOptions</td>
<td>Sets all options for the Rain component.</td>
<td>RainOptions</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetRainOptions</td>
<td>Returns a data structure that contains all options for the Rain component.</td>
<td>None</td>
<td>RainOptions</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------------</td>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>UpdateRain</td>
<td>Updates Lumberyard to use this Rain component as the base for rain effects. The Rain component that is called last takes priority; Lumberyard uses that component.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Request Bus Example Script**

```lua
function example:OnActivate()
    RainComponentRequestBus.Event.SetAmount(self.entityId, 5.0)
    RainComponentRequestBus.Event.Toggle(self.entityId)
end
```

**Random Timed Spawner**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Random Timed Spawner component to spawn a specified dynamic slice. The component spawns the slice at a specified interval (and random interval variation) to a random position inside of a specified volume.

The Random Timed Spawner component requires that the following components are also attached to the entity:

- **Spawner (p. 859)** component – Manages the spawning of the slice.
- **Box Shape (p. 818)** or **Cylinder Shape (p. 818)** component – Defines the volume of possible positions where the entities can spawn.

With these components, you can control the random distribution that determines the random points inside the volume.

Currently, you can specify how entities spawn in random positions with the following random distribution types:

- Normal distribution
- Uniform Real distribution
The **Random Timed Spawner** component is part of the LmbrCentral gem. For more information, see Add modular features and assets with Gems (p. 1064).

**Example**

The following **Random Timed Spawner** component has the distribution type, Uniform Real.

---

**Contents**

- Random Timed Spawner Properties (p. 777)
  - Timing (p. 778)
- EBus Request Bus Interface (p. 778)
  - Enable (p. 778)
  - Disable (p. 779)
  - Toggle (p. 779)
  - IsEnabled (p. 779)
  - SetRandomDistribution (p. 779)
  - GetRandomDistribution (p. 780)
  - SetSpawnDelay (p. 780)
  - GetSpawnDelay (p. 780)
  - SetSpawnDelayVariation (p. 780)
  - GetSpawnDelayVariation (p. 781)
  - Notification Bus Example Script (p. 781)

**Random Timed Spawner Properties**

**Enabled**

Enables slices to spawn with the specified parameters.
Default value: True

**Random Distribution**

Specify the random distribution type when the Random Timed Spawner component creates random spawn positions.

Default value: Uniform Real

Valid values: Normal, Uniform Real

**Timing**

**Spawn Delay**

Time between spawns, in seconds.

If the Enabled parameter is true, a value of 2.0 means that every two seconds after the Random Timed Spawner component is activated, a spawn is triggered.

Default value: 5.0

Valid values: $\infty$ to $\infty$

**Spawn Delay Variation**

Amount of random variation to apply to the spawn delay.

For example, a value of 1.0 and a Spawn Delay value of 2.0 means that the range of possible spawn times is (2.0 +/- 1.0) or between 1.0 and 3.0 seconds.

Default value: 0

Valid values: $\infty$ to $\infty$

**EBus Request Bus Interface**

You can use this EBus to communicate to an entity with a Random Timed Spawner component attached. The EBus is available at game run time and editing and can be accessed from C++, Lua, and the Script Canvas editor.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**Enable**

Starts random spawning.

**Parameters**

None

**Return**

None

**Scriptable**

Yes
**Disable**

Stops random spawning.

**Parameters**

None

**Return**

None

**Scriptable**

Yes

**Toggle**

Toggles random spawning.

**Parameters**

None

**Return**

None

**Scriptable**

Yes

**IsEnabled**

Returns whether the **Random Timed Spawner** component is spawning entities.

**Parameters**

None

**Return**

Type: Boolean

**Scriptable**

Yes

**SetRandomDistribution**

Sets the type of random distribution that determines where the **Random Timed Spawner** component places the entities that spawn.

**Parameters**

Type: AZ::RandomDistributionType

**Return**

None
Scriptable
Yes

GetRandomDistribution
Returns the type of random distribution for the Random Timed Spawner component.

**Note**
In Lua, this value is returned as a number:
0 – Normal distribution
1 – Uniform Real distribution

Parameters
None

Return
Type: AZ::RandomDistributionType

Scriptable
Yes

SetSpawnDelay
Sets the time between entity spawns, in seconds.

Parameters
Type: Double

Return
None

Scriptable
Yes

GetSpawnDelay
Returns the time between entity spawns, in seconds.

Parameters
None

Return
Type: Double

Scriptable
Yes

SetSpawnDelayVariation
Sets the random variation in spawn delay, in seconds.
The **Random Timed Spawner** component uses the specified value to calculate a random value that is added to the **Spawn Delay** parameter.

For example, a **SpawnDelayVariation** value of 1.0 and a **Spawn Delay** value of 2.0 means that the range of possible spawn times is \((2.0 \pm 1.0)\) or between 1.0 and 3.0 seconds.

For more information, see [Timing](p. 778).

**Parameters**

Type: Double

**Return**

None

**Scriptable**

Yes

**GetSpawnDelayVariation**

Returns the random variation in spawn delay, in seconds.

**Parameters**

None

**Return**

Type: Double

**Scriptable**

Yes

**Notification Bus Example Script**

```lua
function example:OnActivate()
    RandomTimedSpawnerComponentRequestBus.Event.SetSpawnDelay(self.entityId, 0.5)
    RandomTimedSpawnerComponentRequestBus.Event.SetSpawnDelayVariation(self.entityId, 0.03)
end
```

**Render to Texture**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](or visit the [AWS Game Tech blog](to learn more).

You can use the **Render to Texture** component to render the scene from a specific camera to a texture. You can use this feature to create rear-view mirrors, security camera screens, and draw 3D models in the viewport.
To enable this component, you must enable the Render to Texture (p. 1197) gem. To enable gems, see Enabling Gems (p. 1064).

Topics

- Adding the Render to Texture Component in a Scene (p. 782)
- Adding a Render Target to a UI Element (p. 783)
- Render to Texture Properties (p. 784)
- EBus Request Bus Interface (p. 794)
- EBus Notification Bus Interface (p. 795)
- Known Limitations (p. 795)

Adding the Render to Texture Component in a Scene

Example

Entity (1) has a Camera (p. 566) component and a Render to Texture component. This entity faces a game object, entity (2). In entity (1), the Render to Texture component specifies a render target. A material file references that render target as its diffuse texture. The material is then added to entity (3), so that the render target appears in the viewport.

To use the Render to Texture component

1. In Lumberyard Editor, create an entity in the viewport. For more information, see Creating an Entity (p. 463).
2. In the Entity Inspector, click Add Component.
3. Select the Camera component, choose Be this camera, and then move the camera entity using the viewport camera controls to look at an object in your level.
4. In the **Entity Inspector**, click **Add Component**.
5. Select the **Render to Texture** component.
6. For **Texture name**, enter a name that starts with the $ character, such as $rendertarget.
   **Note**
   The $ character tells Lumberyard that the texture will not be loaded from disk. If you don't specify the $ character, Lumberyard will try to locate the texture file from disk, which can impact performance.
7. To open the **Material Editor**, choose **Tools, Material Editor** or press M.
8. Click the **Add New Item** icon to create a material, enter a name such as **RenderToTexture**, and then click **Save**.
9. In the **Texture Maps** section, for **Diffuse**, enter the **Texture name** that you specified, such as $rendertarget.
10. In the **Asset Browser**, navigate to **primitive_plane.cgf**, and then select and drag the file into the viewport.
11. Assign the material to the primitive plane entity. In the **Mesh** component, for the **Material override** property, specify the material file that you created, such as **RenderToTexture.mtl**.
   The primitive plane entity now shows the texture from the **Render to Texture** component.

**Adding a Render Target to a UI Element**

You can also specify the render target in an **Image** component for a UI element. For example, you can add the render target to appear in a button or object that a user can select in a UI menu.

For more information, see Visual Components (p. 2936).

**Example**

A camera entity (1) faces the **Render to Texture** entity (2), which has a **Render to Texture** component attached. The **Texture name** is $rendertarget.

![Image of camera and render target](image)

**To add a render target to a UI Image component**

1. Complete steps 1 to 6 in the previous procedure. See Adding the Render to Texture Component in a Scene (p. 782).
2. In Lumberyard Editor, choose **Tools, UI Editor**.
3. In the **UI Editor**, create or open a UI canvas.
4. Right-click the canvas and choose **New, Element from Slice Library, Image.**
5. In the **Image** component, for **SpriteType**, click the drop-down menu, and then select **Render target.**
6. For **Render target name**, enter the same texture name in the **Render to Texture** component, such as `$rendertarget`.

The **UI Image** component displays the render target in the **UI Editor**.

---

### Render to Texture Properties

![Render to Texture Properties](Image)

- **Camera**: 
  - **Texture name**: `$rendertarget`
  - **Max FPS**: 30.0
  - **Width**: 512
  - **Height**: 512
  - **Apply Gamma**: unchecked
  - **Alpha Mode**: **Opaque**
- **Scene Settings**
  - **Enable Ocean**: checked
  - **Enable Terrain**: checked
  - **Enable Vegetation**: checked
  - **Enable Shadows**: checked
  - **GSM LODs**: -1
  - **GSM range**: -1.0
  - **GSM range step**: -1.0
- **Post Effects**
  - **Antialiasing Mode**: **FXAA**
- **Debug**
  - **Display Debug Image**: unchecked
  - **Update in editor**: checked
The **Render to Texture** component has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Camera** | Entity with a Camera component attached to use as the view. If you specify an entity without a Camera component, only the position and orientation will be used from that entity. If you don't specify an entity, this property defaults to the entity to which the Render To Texture component
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture name</td>
<td>the texture to which to render. Follow the Lumberyard convention of using a $ as the first character in the texture name. This is how Lumberyard denotes render targets.</td>
</tr>
<tr>
<td>Maximum FPS</td>
<td>per second at which this view updates.</td>
</tr>
<tr>
<td>Width</td>
<td>of the render texture, in pixels.</td>
</tr>
</tbody>
</table>
Render to Texture

Name

Description

Height

Height of the render texture, in pixels.
**Apply Gamma**

**Description**

Apply Gamma application to the texture. If you want to use this texture with a material inside the main view, leave this property disabled, because gamma will be applied to the entire scene during post-processing.

**Note**

As a best practice, if you are using render targets in the UI, clear the **Apply Gamma**
property and don't select the Render Target sRGB property in the UI Image (p. 2936) component. However, if the texture appears too dark, you can set either property.
**Description**

**Alpha Mode**

Specify the following values:

- **Opaque**
  
  Disables alpha output in the render target.

- **Depth Based**
  
  Uses the scene depth for the alpha channel in the render target.

**Enable Ocean**

Enables drawing the ocean in the render target.

**Enable Terrain**

Enables terrain in the render target.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vegetation</td>
<td>Enables vegetation in the render target.</td>
</tr>
<tr>
<td>Shadows</td>
<td>Enables shadows in the render target.</td>
</tr>
<tr>
<td>GSM LODs</td>
<td>Global shadow map levels of detail (LOD) to use for the render to texture scene. A value of -1 will use the current settings for the e_GsmLodsNum console variable.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GSM range</td>
<td>Global shadow map range to use for the render to texture scene. A value of -1 will use the current settings for the e_GsmRange console variable.</td>
</tr>
</tbody>
</table>
### Description

**GSM Global Shadow Map**

<table>
<thead>
<tr>
<th>range step</th>
<th>use for</th>
<th>render to texture scene.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A value of -1 will use the current settings for the e_GsmRangeStep console variable.

### Antialiasing Mode

Specify the following values:

- **FXAA**
  - Enables fast approximate antialiasing (FXAA) when rendering the scene to the render target.
- **None**
  - Disables antialiasing.
### Render to Texture

**Description**

**Displays Debug Image**

Images in Lumberyard Editor only.

**Updates in editor**

Render target outside of game mode while in Lumberyard Editor.

---

### EBus Request Bus Interface

You can use the event bus (EBus) interface to communicate with other components in your game.

For more information, see [Working with the Event Bus (EBus) system](#) (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetTextureResourceId</td>
<td>Gets the resource ID of the render target. A value of -1 is an invalid render target.</td>
<td>None</td>
<td>integer</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| SetAlphaMode          | Sets the alpha mode for the render target.                                  | 0 = Disabled  
1 = Opaque  
2 = Depth based | None      |         | Yes        |
| SetCamera             | Sets the entity with a **Camera** component to use for rendering.           | EntityID  | None    | Yes        |
| SetEnabled            | Enables or disables rendering to texture.                                  | 0 = Disable  
1 = Enable   | None    |         | Yes        |
### Request Name | Description | Parameter | Return | Scriptable
--- | --- | --- | --- | ---
SetMaxFPS | Sets the maximum rate at which this render target updates, in frames per second. | FPS | None | Yes
SetWriteGamma | Enables or disables gamma application. | 0 = Disable | None | Yes
1 = Enable

### EBus Notification Bus Interface
You can use the event bus (EBus) interface to communicate with other components in your game.

For more information, see Working with the Event Bus (EBus) system (p. 1851).

### Known Limitations
The **Render to Texture** component supports DirectX 11 for Windows.

**Prevent Graphical Artifacts When Render to Texture is Active with the Following CVARs**

Forces the object LOD to update at the beginning of the frame instead of the end of the previous frame. If you don’t set this console variable, you may see LOD flickering.

For more information about setting console variables, see Using the Console Window (p. 210).

- `e_LodForceUpdate=1`

Disables precaching of the streaming system due to common large jumps in the camera position when the render to texture feature is active.

- `e_AutoPrecacheCameraJumpDist=0`
Enables subpixel morphological antialiasing (SMAA) instead of temporal antialiasing (TAA).

\[ r_{\text{AntialiasingMode}}=2 \]

Disables volumetric fog.

\[ e_{\text{volumetricFog}}=0 \]

Disables multi-GPU mode.

\[ r_{\text{MultiGPU}}=0 \]

Disables sparse voxel octree global illumination (SVOGI gem).

\[ e_{\text{GI}}=0 \]

You might see flickering, black or blurry textures if the streaming system thrashes while the `Render to Texture` component is active. This can occur in a scene that uses a large amount of texture memory.

To fix this, increase the value for `r_TexturesStreamPoolSize`, or set the `Max FPS` property to 0 so that the component updates each frame. This might help the streaming system balance texture priorities.

**Tip**

To verify if the graphics artifacts are a result of streaming thrashing, set `r_TexturesStreamingDebug` to 2. This console variable shows the textures that are streamed in and the memory usage.

### Unsupported Features in the Render Target

The following features are not supported for the `Render to Texture` component because they don't work with multiple cameras.

- TAA and SMAA
- Lens flares
- Sun shafts
- Volumetric fog
- CBuffer (constant buffer) occlusion (vis areas are supported)
- Motion blur
- Merged mesh vegetation
- Shadows cache

**River**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Use the River component to carve riverbeds into the terrain and fill them with moving water. The River component uses 2D materials to create the illusion of flowing water. Optimally, you should use this component to create flat rivers along level terrain. To create rivers that run downhill, you can create a series of rivers separated by occasional waterfalls.

The River component requires the Spline (p. 866) component to shape its path along the x, y, and z axes. After you place a river, you can edit the points in the river's spline.

To use the River component, you must enable (p. 1064) the Roads and Rivers gem. For more information, see Enabling Gems (p. 1064).

Topics
- River Properties (p. 798)
- Working with the River Component (p. 800)
- River Request Bus Interface (p. 805)
See the following River component properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global width</td>
<td>Sets the width of the river.</td>
</tr>
<tr>
<td>Per-Vertex Width Modifiers</td>
<td>Modifies the width at a specific vertex that is added to or subtracted from the Global width. For more information, see Modifying the River Width (p. 803).</td>
</tr>
<tr>
<td>Elements</td>
<td>The river's vertices or points. Specify a positive or negative value for each vertex.</td>
</tr>
<tr>
<td>Segment length</td>
<td>Length of each segment. Smaller segment lengths increase the polycount used for the river curvature. To create a river with smooth corners, specify lower values.</td>
</tr>
<tr>
<td>Tile length</td>
<td>Length of the river texture. Adjust this parameter with Segment Length to avoid stretching textures.</td>
</tr>
<tr>
<td>Sort priority</td>
<td>Priority that you can set if the river intersects with another river.</td>
</tr>
<tr>
<td>View distance multiplier</td>
<td>Distance at which the river is visible. The default value is 1. A higher value indicates a longer visibility distance.</td>
</tr>
<tr>
<td>Minimum spec</td>
<td>Specifies the minimum system configuration level for the effect. If the configuration is lower than the set value, the river will not appear. Choose from Low, Medium, High, VeryHigh, and Never. Default value: Low</td>
</tr>
<tr>
<td>Tile width</td>
<td>Width of the river texture.</td>
</tr>
<tr>
<td>Depth</td>
<td>Depth of the river.</td>
</tr>
<tr>
<td>River material</td>
<td>Material file for the river. For more information, see Setting River Material Properties (p. 804).</td>
</tr>
<tr>
<td>Cap at Depth</td>
<td>If selected, limit fog rendering to the river's bottom.</td>
</tr>
<tr>
<td></td>
<td>If deselected, fog continues to render below the specified depth of the river.</td>
</tr>
<tr>
<td>Density</td>
<td>Specifies how dense the fog appears. Specify higher values for thicker fog.</td>
</tr>
<tr>
<td>Color</td>
<td>Color of the fog.</td>
</tr>
<tr>
<td>Affected by Sun</td>
<td>If selected, the value for the Sun color parameter in the Time of Day Editor (p. 1266) affects fog color of the river.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Shadowing</td>
<td>Sets the shadow darkness on the surface of the river.</td>
</tr>
<tr>
<td></td>
<td>To enable this parameter, set the console variable r_FogShadowsWater to 1. For more information, see Using the Console Window (p. 210).</td>
</tr>
<tr>
<td>Caustics - Enabled</td>
<td>Enables caustics – Light rays that are reflected or refracted by the water's surface.</td>
</tr>
<tr>
<td>Intensity</td>
<td>Intensity of normals during caustics generation.</td>
</tr>
<tr>
<td>Height</td>
<td>Distance above water's surface that caustics are visible.</td>
</tr>
<tr>
<td>Tiling</td>
<td>Tiling of normals during caustic generation.</td>
</tr>
<tr>
<td>Physics - Enabled</td>
<td>Bind river with CryPhysics.</td>
</tr>
<tr>
<td>Speed</td>
<td>Defines how fast physicalized objects are moved along the river. Negative values move objects in the opposite direction.</td>
</tr>
<tr>
<td>Border width</td>
<td>Specifies the width of the slope on the borders of the river.</td>
</tr>
<tr>
<td>Embankment height</td>
<td>Defines the height of the slopes on the borders of the river. If the river is at ground level, specify an embankment height to make the river appear to flow.</td>
</tr>
<tr>
<td>Depth of the river bed</td>
<td>Defines the depth of the ditch beneath the river.</td>
</tr>
<tr>
<td>River bed width offset</td>
<td>Defines the width of the riverbed relative to the river's width. A value of 0 makes the riverbed the same width as the river. A value of -1 makes the riverbed one meter narrower, and so on.</td>
</tr>
<tr>
<td>Erase width</td>
<td>Distance from the river edges in which the vegetation should be removed. Set the parameter to 0 to remove only vegetation that is on the river. Specify positive values to remove vegetation in a wider border from the river's edge. Specify negative values to leave some vegetation within the borders of the river.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Erasing River Vegetation (p. 804).</td>
</tr>
<tr>
<td>Erase variance</td>
<td>Randomizes the removal of vegetation so that the edge looks more natural.</td>
</tr>
</tbody>
</table>

**Working with the River Component**

See the following procedures to create a river and modify its appearance.
Creating a River

To create a realistic river, use the River component's Carve River Bed tool to prepare the surrounding terrain. This tool carves out a riverbed and creates riverbanks on the edges of the river that are higher than the river's water level.

You can also paint the river bottom with a different texture than the surrounding area. This is useful if the water has translucent properties. For more information on painting the terrain, see Painting Terrain Texture Layers (p. 1234).

To create a river

1. Create an entity. (p. 463)
2. Add (p. 479) the River component.
3. Add the Spline (p. 866) component.

   The Spline component defines the path and shape of the river. For information about how to add, remove, and edit individual points, see the Spline (p. 866) component.

4. Under Terrain Editing, specify values for the following parameters:
   - **Border width** – Sets the width of the slope on the river's sides.
   - **Embankment height** – Defines the height of the hills on the river edges. If the river is at ground level, this parameter makes the river appear to flow in the riverbed.
   - **Depth of the river bed** – Defines the depth of riverbed.
   - **Riverbed width offset** – Defines the width of the riverbed in relation to the river's width. For example, a value of 0 makes the riverbed the same width as the river; –1 makes the riverbed one meter narrower than the river; 1 makes the riverbed one meter wider than the river.

5. Click Carve River Bed.

Example

A river on the ground before a riverbed has been carved out.
A river with a **Border width** of 5.0, **Embankment** of 1.0, **Depth** of 2.0, and **River bed offset** of -1.

You can also lower the river below the ground level and carve the riverbed. This achieves a different type of terrain deformation.

A river with a **Border width** of 10.0, **Embankment** of 3.0, **Depth** of 2.0, and **River bed offset** of -1.

This river is positioned below ground level.
Modifying the River Width

You can use the **Global width** parameter to adjust the river's width. This parameter modifies all the points on the river.

For more refined control over the river's width, you can adjust the individual points with the **Per-Vertex Width Modifiers** parameter.

![Per-Vertex Width Modifiers](image)

**To adjust the width of individual points**

- In the **River** component, specify a negative or positive number for the element that you want to modify.

  Negative numbers subtract from the **Global Width** parameter, while positive numbers add to it. A value of 0 means that point is at global width.

**Example**

The global width of this river is 2, and the width at the selected point is set to 5. That means the total width at the selected point is 7.
Erasing River Vegetation

You can erase vegetation around the river's borders.

To erase vegetation

1. In the River component, for Vegetation Editing, set the Erase width. This defines the amount of vegetation to erase at the river's edges.
2. Set the Erase variance. This parameter randomizes the removal of vegetation so that the edge looks more natural.

Setting River Material Properties

The road material must use the WaterVolume Shader (p. 1719).

To set the river material

1. Press M or navigate to Tools, Material Editor
2. Select the defaultRiver or other appropriate river material.
For information about the WaterVolume Shader properties, see WaterVolume Shader (p. 1719).

**River Request Bus Interface**

The River component uses two event bus (EBus) interfaces, the RiverRequestBus and the RoadsAndRiversGeometryRequestsBus, which is shared with the Road component.

For information about the RoadsAndRiversGeometryRequestsBus, see RoadsAndRiversGeometryRequestsBus Interface (p. 817).

Use the following request functions with the RoadsAndRiversGeometryRequestsBus to communicate with other components of your game.

For more information, see Working with the Event Bus (EBus) system (p. 1851).
<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetWaterVolumeDepth</td>
<td>Sets the water volume depth of the river in meters.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterVolumeDepth</td>
<td>Returns the setting for the water volume depth of the river in meters.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetTileWidth</td>
<td>Sets the river's tile width in meters.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetTileWidth</td>
<td>Returns the setting for the river's tile width in meters.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterCapFogAtVolumeDepth</td>
<td>Sets whether to enable capping the fog at the depth of the river.</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterCapFogAtVolumeDepth</td>
<td>Returns whether fog setting for the Cap at Depth parameter is enabled.</td>
<td>None</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterFogDensity</td>
<td>Sets the density of the fog in the water.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterFogDensity</td>
<td>Returns the setting for the density of the fog in the water.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetFogColor</td>
<td>Sets the color of the fog in the water.</td>
<td>AZ::Color</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetFogColor</td>
<td>Returns the setting for the color of the fog in the water.</td>
<td>None</td>
<td>AZ::Color</td>
<td>Yes</td>
</tr>
<tr>
<td>SetFogColorAffectedBySun</td>
<td>Sets whether the sun affects the fog color.</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetFogColorAffectedBySun</td>
<td>Returns whether the fog setting for the Affected by Sun parameter is enabled.</td>
<td>None</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>SetWaterFogShadowing</td>
<td>Sets the shadow's darkness on the surface of the river.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterFogShadowing</td>
<td>Returns the setting for the shadow's darkness on the surface of the river.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterCaustics</td>
<td>Sets whether to enable caustics.</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterCaustics</td>
<td>Returns whether caustic setting is enabled.</td>
<td>None</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterCausticIntensity</td>
<td>Sets the intensity of normals during caustic generation.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterCausticIntensity</td>
<td>Returns the setting for the caustic Intensity parameter.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterCausticHeight</td>
<td>Sets the distance above the water's surface that caustics are visible.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterCausticHeight</td>
<td>Returns the setting for the distance above the water's surface that caustics are visible.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterCausticTiling</td>
<td>Sets the tiling of normals during caustics generation.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterCausticTiling</td>
<td>Returns the setting for the tiling of normals during caustics generation.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetPhysicalize</td>
<td>Sets whether to enable Physics (binds river with CryPhysics).</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>GetPhysicalize</td>
<td>Returns the setting for whether Physics is enabled.</td>
<td>None</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>SetWaterStreamSpeed</td>
<td>Sets the speed at which physicalized objects are moved along the river. Negative values move objects in the opposite direction.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetWaterStreamSpeed</td>
<td>Returns the setting for the speed at which physicalized objects are moved along the river.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Road**

This feature is in *preview* release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Road component to create paths in Lumberyard Editor. You can create roads that follow the curvature of existing terrain by applying a texture over the terrain texture. You can also use the Align heightmap feature with the Road component to shape the terrain to the height and curvature of the road you placed.

To enable the Road component, you must enable the Roads and Rivers gem. For more information, see Add modular features and assets with Gems (p. 1064).

The Road requires the Spline (p. 866) component to shape its path along the X, Y, and Z axes. After you place a road, you can edit the points in the road's spline.

**Note**

You can rebuild roads at runtime by modifying the spline. However, this method can be slow. We recommended that you avoid modifying the spline continuously at runtime.

**Topics**

- Road Properties (p. 809)
- Creating a Road (p. 810)
- Modifying Road Width (p. 811)
- Setting Road Material Properties (p. 812)
- Using Roads to Modify Terrain (p. 813)
Road Properties

See the following Road properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global width</td>
<td>Width of the road.</td>
</tr>
<tr>
<td>Per-Vertex Width Modifiers</td>
<td>Width at a specific vertex that is added to or subtracted from the global width. For more information, see Modifying Road Width (p. 811).</td>
</tr>
<tr>
<td>Elements</td>
<td>The road's vertices or points. Specify a positive or negative number for each vertex.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Segment length</td>
<td>Length of each segment. Smaller segment lengths increase the polycount used for the road surface. To create a road with smooth corners, use smaller segment lengths.</td>
</tr>
<tr>
<td>Tile length</td>
<td>Length of the road texture. Adjust Tile length with Segment length to avoid stretching textures.</td>
</tr>
<tr>
<td>Sort priority</td>
<td>Priority that you can use if the road is drawn over or under another road.</td>
</tr>
<tr>
<td>View distance multiplier</td>
<td>Distance at which the road is visible. A higher number indicates a longer visibility distance. The default value is 1.</td>
</tr>
<tr>
<td>Minimum spec</td>
<td>Specifies the minimum system configuration level for the effect. If the configuration is lower than the set value, the road will not appear. Choose from Low, Medium, High, VeryHigh, and Never. Default value: Low</td>
</tr>
<tr>
<td>Road material</td>
<td>Sets the material for the road. For more information, see Setting Road Material Properties (p. 812).</td>
</tr>
<tr>
<td>Ignore terrain holes</td>
<td>If enabled, the road texture is rendered over terrain holes.</td>
</tr>
<tr>
<td>Border width</td>
<td>Width of the border extending from the sides of the road. Use with the Align heightmap (p. 813) feature.</td>
</tr>
<tr>
<td>Erase width</td>
<td>Distance from the road edges in which the vegetation should be removed. Specifying a value of 0 removes only vegetation that is on the road. Specify positive values to remove vegetation in a wider border from the road’s edge. Specify negative values to leave some vegetation within the borders of the road. For more information, see Erasing Vegetation (p. 815).</td>
</tr>
<tr>
<td>Erase variance</td>
<td>Randomizes the removal of vegetation so that the road edges appear more natural.</td>
</tr>
</tbody>
</table>

**Creating a Road**

**To create a road**

1. Create an entity. (p. 463)
2. Add (p. 479) the Road component.
3. Add the Spline (p. 866) component.
The **Spline** component defines the path and shape of the road. For information about how to add, remove, and edit individual points, see [Spline](p. 866).

The road is created with the default material **defaultRoad**. The **Spline** component defaults to the **Linear Spline Type** and contains four vertices (0, 1, 2, 3). For a road that curves smoothly, specify the [Bezier](p. 871) **Spline Type**.

### Modifying Road Width

You can modify the road's width by adjusting the road's **Global width** property. Adjusting this property modifies all the points on the road.

For more granular control over the road's width, you can adjust individual points with the **Per-Vertex Width Modifiers** property.

**To adjust the width of individual points**

- In the **Road** component properties, enter a negative or positive number for the property that you want to modify.

  Negative numbers subtract from the **Global width** property, while positive numbers add to it. A value of 0 means that point is at global width.

  **Example**

  The global width of this road is 2, and the width at the selected point is set to 5. The total width at the selected point is 7.
Setting Road Material Properties

Roads are Decals (p. 585) placed along a spline. This makes the material setup of a road similar to that of decals.

The road material must use the Illum Shader (p. 1697) and should also enable the Decal and Vertex Colors parameters in the Shader Generation Params.

To open the Material Editor

1. In Lumberyard Editor, choose Tools, Material Editor or press M.
2. Select the road material and expand the Shader Generation Params, and select the following parameters:

   **Vertex Colors**

   Activates a blend fadeout at each end of the road. This is a 100% to 0% fadeout over the length of the final step. The textures and alpha channel have no influence on the fadeout.

   **Decal**

   Enables alpha blending on the sides of the road, which comes from the diffuse texture's alpha channel.
**Example**

In the left image, **Vertex Colors** is enabled. In the right image, the parameter is disabled.

**Using Roads to Modify Terrain**

You can use roads to modify the shape of the terrain.
To modify the terrain with roads

1. Create a road. (p. 810)
2. Manipulate the individual vertices to shape the road. For more information, see Spline (p. 866).
3. In the Road component's properties, under Terrain Editing, set the Border width. This parameter defines the width of the slope at the edges of the road.
4. Click Align heightmap.

Example

The following image shows a road that has been created and shaped, but not yet aligned.

Example

The following image shows the road and terrain after alignment with a Border width of 5.

Example

The following image shows the road and terrain after alignment with a Border width of 20.
Erasing Vegetation

You can erase vegetation around the road's borders.

**To erase vegetation**

1. In the Road component's properties, under *Vegetation Editing*, specify the *Erase width*. This parameter defines the amount of vegetation to erase at the road's edges.

2. Specify the *Erase variance*. This parameter randomizes the removal of vegetation, so that the edge looks more natural.

**Example**

The following image shows the road with no vegetation erased.

**Example**

The following image shows the road with *Erase width* set to 0.
Example

The following image shows the road with **Erase width** set to 1 and **Erase variance** set to 5.

RoadRequestBus Interface

Use the following request functions with the RoadRequestBus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rebuild</td>
<td>Triggers full rebuild of the road object, including geometry and render node generation.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>SetIgnoreTerrainHoles</td>
<td>Sets whether to allow the road</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### RoadsAndRiversGeometryRequestsBus Interface

Use the following request functions with the `RoadsAndRiversGeometryRequestsBus` EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Working with the Event Bus (EBus) system](#) (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SetVariableWidth</code></td>
<td>Sets the variable width along the river using the index of spline.</td>
<td>AZ::u32, float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SetGlobalWidth</code></td>
<td>Sets the uniform global width along the road or river. This value is added to the variable width specified by the spline width attribute. If no width attribute specified, the width of the spline geometry is uniform and equal to this parameter.</td>
<td>float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SetTileLength</code></td>
<td>Sets the tile length, which is the texture mapping scale along the geometry spline.</td>
<td>float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SetDesiredGranularity</code></td>
<td>Sets the desired granularity, which is the size of the polygon along the road or river. Lower numbers generate more polygons.</td>
<td>float</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the **Script Canvas** component to add a script to an entity.

**Script Canvas Component Properties**

The **Script Canvas** component has the following properties:

**Script Canvas Asset**

Specify the script file that you want to add to the component.

**Shapes: Cylinder, Capsule, Disk, Box, Sphere, Compound**

Add shape components to entities that have other components that require shapes.

For example, add shape components to your entities to create volumes for the following:

- Create a trigger volume using the *Trigger Area (p. 890)* component. Entities that enter the volume trigger an action.
- Create a collision volume using the *PhysX Collider (p. 735)* component. Entities that collide with the collision volume respond to the impact, such as slowing down or bouncing back.
- Define reverb or echo effects using the *Audio Environment (p. 552)* component.
- Create an audio area using the *Audio Trigger (p. 559)* component so that a specific sound plays in that area of the level.

**Note**

You can add only one shape component to an entity. If you need more than one shape on an entity, create child entities.

Each shape component provides a generic *ShapeService* that exposes functionality common to all shapes. Each shape also provides a more specific service, such as *BoxShapeService* or *SphereShapeService*.

The **Shape** component includes the following shapes and properties:

**Topics**

- Scaling Shapes (p. 819)
- Box Shape Component Properties (p. 820)
- Capsule Shape Component Properties (p. 821)
• Compound Shape Component Properties (p. 822)
• Cylinder Shape Component Properties (p. 822)
• Disk Shape Component Properties (p. 823)
• Sphere Shape Component Properties (p. 824)
• Request EBus Interface (p. 824)

Scaling Shapes

As a best practice, when you scale a shape, adjust the shape component's properties, such as its Dimensions, Height, and Radius.

Tip
Modify the shape component's properties instead using the Scale tool (p. 201), which changes the entity's Transform (p. 878) component.

Example

See the following examples for the different scaling:

1. In nonuniform scaling, adjusting an entity's transform scale leads to different values for X, Y, and Z.
2. In uniform scaling, X, Y, and Z have the same value.
3. In uniform normalized scaling, all of the scale values are 1.

If an entity has been scaled nonuniformly, rendering and intersection tests use the largest component of the scale vector. Although this can work, the best practice is to keep a uniform and, ideally, normalized (1, 1, 1) transform scale, as shown in the third example.
Box Shape Component Properties

The **Box Shape** component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the box shape in game mode in Lumberyard Editor.</td>
</tr>
<tr>
<td></td>
<td>To enter game mode, press <strong>Ctrl+G</strong></td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Dimensions</td>
<td>Defines dimensions of the shape along all three axes in local space.</td>
</tr>
<tr>
<td></td>
<td>If the Transform (p. 878) component applies a scale, these dimensions are multiplied by the same value.</td>
</tr>
<tr>
<td>Edit</td>
<td>Choose <strong>Edit</strong>, and the component is locked for editing. For more information, see Editing Components in the Viewport (p. 483).</td>
</tr>
</tbody>
</table>

The **Box Shape** component also has its own Component Mode with several linear manipulators for each axis.
Capsule Shape Component Properties

The Capsule Shape component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the shape in game mode.</td>
</tr>
<tr>
<td></td>
<td>To enter game mode, press Ctrl + G.</td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Height</td>
<td>The end-to-end height of the capsule, including the cylinder and both caps.</td>
</tr>
</tbody>
</table>
Shapes: Cylinder, Capsule, Disk, Box, Sphere, Compound

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td>The radius of the capsule. If the Transform (p. 878) component applies a scale, these dimensions are multiplied by the same value.</td>
</tr>
</tbody>
</table>

### Compound Shape Component Properties

With the **Compound Shape** component, you can combine shapes to create a complex object to generate a physics collider, trigger shape, or any other application of shapes. The individual shapes might or might not be children of the entity with the **Compound Shape** component.

**Note**

- To move the shapes together with the compound shape, set them as child entities.
- To move the shapes independent of the entity with the compound shape, don’t set them as child entities.
- For EBus (event bus) requests, compound shapes service the full shape component bus. However, each individual shape that you add increases the cost of requests such as `IsPointInside`.

See the following **Compound Shape** properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child Shape Entities</td>
<td>Specifies the entities and their shapes that make up the compound shape. These entities might or might not be children of the entity with the compound shape.</td>
</tr>
</tbody>
</table>

### Cylinder Shape Component Properties

The **Cylinder Shape** component has the following properties.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn’t selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the shape in game mode in Lumberyard Editor.</td>
</tr>
<tr>
<td></td>
<td>To enter game mode, press Ctrl + G.</td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the cylinder. If the Transform (p. 878) component applies a scale, these dimensions are multiplied by the same value.</td>
</tr>
<tr>
<td>Radius</td>
<td>The radius of the cylinder. If the Transform (p. 878) component applies a scale, these dimensions are multiplied by the same value.</td>
</tr>
</tbody>
</table>

**Disk Shape Component Properties**

![Disk Shape Component Properties](image)

See the following Disk Shape properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn’t selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the shape in game mode in Lumberyard Editor.</td>
</tr>
<tr>
<td></td>
<td>To enter game mode, press Ctrl + G.</td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Radius</td>
<td>The radius of the disk.</td>
</tr>
<tr>
<td></td>
<td>If the Transform (p. 878) component applies a scale, these dimensions are multiplied by the same value.</td>
</tr>
</tbody>
</table>
Sphere Shape Component Properties

See the following Sphere Shape properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the shape in game mode in Lumberyard Editor.</td>
</tr>
<tr>
<td></td>
<td>To enter game mode, press Ctrl + G.</td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Radius</td>
<td>The radius of the sphere. If the Transform component applies a scale, these dimensions are multiplied by the same value.</td>
</tr>
</tbody>
</table>

Request EBus Interface

All shape components provide access to two separate request buses. The first bus is the ShapeComponentRequestsBus (p. 824) that returns general information about the shape component. The second bus is the ShapeNameComponentRequestsBus (p. 825) that returns property configurations for the specified shape.

Use the following request functions with the event bus interface to communicate with other components in your game.

For more information, see Working with the Event Bus (EBus) system (p. 1851).

ShapeComponentRequestsBus

The following table describes the ShapeComponentRequestsBus.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistanceFromPoint</td>
<td>Returns the minimum distance between a specified point and the shape.</td>
<td>point - Vector3 point to calculate distance from.</td>
<td>Distance from point to shape. Type: Float</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Shapes: Cylinder, Capsule, Disk, Box, Sphere, Compound

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DistanceSquaredFromPoint</strong></td>
<td>Returns the minimum squared distance between a specified point and the shape.</td>
<td>point – Vector3 point to calculate square distance from.</td>
<td>Square distance from point to shape. Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>GetEncompassingAabb</strong></td>
<td>Returns an AABB that encompasses the entire shape.</td>
<td>None</td>
<td>AZ::Aabb that encompasses the shape.</td>
<td>No</td>
</tr>
<tr>
<td><strong>GetShapeType</strong></td>
<td>Returns the specified shape type for the component.</td>
<td>None</td>
<td>AZ::Crc32(shape)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>For example: AZ::Crc32(&quot;Box&quot;)</td>
<td></td>
<td></td>
<td>AZ::Crc32(&quot;Sphere&quot;)</td>
</tr>
<tr>
<td><strong>IsPointInside</strong></td>
<td>Checks whether a specified point is inside or outside a shape.</td>
<td>point – Vector3 point to check.</td>
<td>Indicates whether the point is inside or outside the shape. Type: Boolean</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**ComponentRequestsBus**

Each shape component has its own specific event bus for accessing that particular shape. The buses have similar functions and only differ for the requested shape type.

**BoxShapeComponentRequestsBus**

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GetBoxConfiguration</strong></td>
<td>Returns the configuration of the BoxShape.</td>
<td>None</td>
<td>BoxShapeConfiguration object that contains the configuration for the box shape.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**CapsuleShapeComponentRequestsBus**

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GetCapsuleConfiguration</strong></td>
<td>Returns the configuration</td>
<td>None</td>
<td>CapsuleShapeConfiguration object that contains the configuration</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameter</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>of the CapsuleShape.</td>
<td></td>
<td>configuration for the capsule shape.</td>
<td></td>
</tr>
</tbody>
</table>

**CompoundShapeComponentRequestsBus**

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCompoundShapeConfiguration</td>
<td>Returns the configuration of the CompoundShape.</td>
<td>None</td>
<td>CompoundShapeConfiguration object that contains the configuration for the compound shape.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**CylinderShapeComponentRequestsBus**

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCylinderConfiguration</td>
<td>Returns the configuration of the CylinderShape.</td>
<td>None</td>
<td>CylinderShapeConfiguration object that contains configuration for the cylinder shape.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**SphereShapeComponentRequestsBus**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSphereConfiguration</td>
<td>Returns the configuration of the SphereShape.</td>
<td>None</td>
<td>SphereShapeConfiguration object that contains the configuration for the sphere shape.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**ShapeComponentNotificationsBus**

All shapes share a single notification bus named ShapeComponentNotificationsBus.

<table>
<thead>
<tr>
<th>Notification Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnShapeChanged</td>
<td>Notifies listeners that the shape component has been updated.</td>
<td>ShapeChangeReasons – Indicates whether the shape was updated from a transform change or a shape parameter change.</td>
<td>Void</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Simple Motion**

This feature is in preview release and is subject to change.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Simple Motion component to play a motion without an animation graph. Add this component to the Actor (p. 536) component to use a single motion for your actor. For complex motions, see the AnimGraph (p. 545) component.

For creating cinematics with the Simple Motion component, see Creating Character Animations with the Simple Motion Component (p. 1622).

**Simple Motion Component Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preview in Editor</td>
<td>Plays the motion in Lumberyard Editor.</td>
</tr>
<tr>
<td>Motion</td>
<td>Specifies the motion that you want the actor to play.</td>
</tr>
<tr>
<td>Loop motion</td>
<td>Runs the animation continuously.</td>
</tr>
<tr>
<td>Retarget motion</td>
<td>Allows motion that was created with an actor that was configured with specific bone lengths to be played on another actor with different bone lengths. When applied, the motion does not affect bone lengths. The skeleton must follow the same hierarchy and the bone names must be identical to work properly.</td>
</tr>
<tr>
<td>Reverse motion</td>
<td>Runs the animation in reverse.</td>
</tr>
<tr>
<td>Mirror motion</td>
<td>Mirrors the animation of the character's body parts. For example, if the actor kicks with the right leg while the left leg is planted, the mirror effect causes the left leg to kick while the right leg is planted.</td>
</tr>
<tr>
<td>Play speed</td>
<td>Specifies the rate at which the motion is played.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Blend in time</td>
<td>Specifies the blend in time for the motion, in seconds. You can set this parameter for a motion that you want to start and that is blending from a previous motion.</td>
</tr>
<tr>
<td>Blend out time</td>
<td>Specifies the blend out time for the motion, in seconds. You can set this parameter for a motion that is currently playing and that will blend into the next motion.</td>
</tr>
</tbody>
</table>

**Simple State**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Simple State** component provides a simple state machine. Each state is represented by a name and zero or more entities. The entities are activated upon entering the state and deactivated upon exiting it. A simple state component may be in NullState, which means no state is active.
Simple State Component Properties

The Simple State component has the following properties:

**Initial state**

The active state when the simple state component is first activated.

**Reset on activate**

If selected, simple state returns to the configured initial state when activated, and not the state held before deactivating.

**States**

The list of states on this simple state component.

**State ([0], [1], [2], etc)**

Includes a name for the state and a set of entities that are activated when the state is entered and deactivated when the state is exited.

**Name**

The name of this state. Indicates the state to which to transition on the SetState API.

**Entities**

List of the entities referenced by this state.

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**SetState**

Sets the active state to the named state.

**Parameters**

- **stateName**

**EBus Notification Bus Interface**

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**OnStateChanged**

Notifies that the state has changed from state **oldName** to state **newName**.

**Parameters**

- **oldName**
The following is an example of script using the **Request Bus Interface**.

```lua
local simplestateexample = {
    Properties = {
        TransitionInterval = 1.0,
        States = {"Houses", "Nope", "Lamps", "Tree", "HouseAndTree", "NoState"},
    }
}

function simplestateexample:OnActivate()
    self.TransitionCountDown = self.Properties.TransitionInterval
    self.StateIdx = 0
    self.tickBusHandler = TickBus.Connect(self)
    self.stateChangedHandler = SimpleStateComponentNotificationBus.Connect(self, self.entityId)
    Debug.Log("SimpleStateComponent activated for entity: " .. tostring(self.entityId.id))
end

function simplestateexample:OnDeactivate()
    self.tickBusHandler:Disconnect()
    self.stateChangedHandler:Disconnect()
end

function simplestateexample:OnTick(deltaTime, timePoint)
    self.TransitionCountDown = self.TransitionCountDown - deltaTime
    if (self.TransitionCountDown < 0.0) then
        self.StateIdx = (self.StateIdx + 1) % table.getn(self.Properties.States)
        self.TransitionCountDown = self.Properties.TransitionInterval
    end
end

function simplestateexample:OnStateChanged(oldState, newState)
    Debug.Log("Old State: " .. (oldState or "NullState") .. " => New State: " .. (newState or "NullState"))
end

return simplestateexample
```

**Sky Cloud**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Sky Cloud** component creates realistic and detailed cloud effects in your game levels. To use the **Sky Cloud** component, you must enable the **Sky Clouds** gem in your project. For more information, see Add modular features and assets with Gems.
For information about cloud shading settings, placing distance clouds, and adding cloud shadows, see *Adding Clouds* (p. 1293).

**Note**
The *Sky Cloud* component replaces *legacy clouds*.

**Example Sky Cloud Component**

Lumberyard features two variations of cloud rendering:

**Common clouds**

This sprite-based rendering path uses simple shading to optimize performance. Close up, these clouds appear in 3D but once the clouds are a certain configurable distance from the viewer, an imposter (p. 3283) is substituted.

**Volumetric clouds**

This implementation uses advanced shading to create realistic voxelized (p. 3286) 3D clouds but is slower to render.

With the *Sky Cloud* component, you can do the following:

- Randomly generate common and volumetric clouds in controlled areas.
- Control where clouds generate by using entities with *Box Shape* (p. 818) components.
- Define the area in which clouds can move with a loop box.
- Programmatically manage *Sky Cloud* component properties with the Script Canvas editor, Lua scripting, and the Track Editor.
- Create and use slices that include cloud components.

The following procedure demonstrates how to create a common or volumetric cloud with the *Sky Cloud* component. After you create a cloud, you can customize its appearance and movement using the procedures in this chapter.
Creating a Cloud with the Sky Cloud Component

To create a cloud

1. Create an entity. (p. 463)
2. In the Entity Outliner, select the new entity and in the Entity Inspector, click Add Component and then click the Sky Cloud component.

3. Click Add Required Component to add the Box Shape (p. 818) component.

4. In the Entity Inspector, at the bottom of the Sky Cloud component properties, click Generate. This generates a common cloud that uses sprite-based shading in the viewport.
5. To turn the common cloud into a volumetric cloud, select the Volumetric Rendering (p. 835) property.
Sky Cloud Component Properties

The Sky Cloud component properties are grouped into the following categories. See the individual sections for detailed descriptions for the available parameters.

Cloud material

Sets the material for common cloud rendering (p. 1653). The default material is `baseclouds.mtl`. To select a different material, click (...) and choose a material based on the common cloud shader.

Volumetric Rendering (p. 835)

Enables volumetric cloud rendering (p. 1714) and specifies the cloud's material and density.

Movement (p. 835)

Defines how the clouds move over time.

Display (p. 835)

Controls which visual aids are displayed when viewing the cloud in Lumberyard Editor.
Generation (p. 836)
Defines parameters for cloud generation when you click Generate.

Volumetric Rendering Parameters
When you enable Volumetric Rendering, you create a realistic-looking volumetric cloud that is voxelized (p. 3286) and uses the Volume Object shader (p. 1714).

When Volumetric Rendering is not enabled, you create a sprite-based common cloud, which uses the Common Cloud shader (p. 1653).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>If selected, this cloud is drawn with volumetric rendering. If not selected, the common cloud shader is used.</td>
</tr>
<tr>
<td>Volume material</td>
<td>Volume Object (p. 1714) based material used for rendering volumetric clouds. Volumetric rendering must be enabled for this parameter. The default material is volumeClouds.mtl.</td>
</tr>
<tr>
<td>Density</td>
<td>Defines the volumetric cloud's density. Volumetric rendering must be enabled for this parameter.</td>
</tr>
</tbody>
</table>

Movement Parameters
The Movement parameters define how the cloud moves within its loop box (p. 836).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoMove</td>
<td>If selected, the cloud moves on its own.</td>
</tr>
<tr>
<td>Velocity</td>
<td>Velocity in meters per second and cloud movement in the X, Y, and Z axes.</td>
</tr>
<tr>
<td>FadeDistance</td>
<td>Distance in meters from the loop box edge. Defines where the cloud begins to fade out before wrapping around to the other side.</td>
</tr>
</tbody>
</table>

Display Parameters
Enable Display parameters to help visualize the cloud while editing.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Spheres</td>
<td>Displays a sphere for each particle. Enable this parameter to visualize the overall volume of the cloud.</td>
</tr>
<tr>
<td>Display Volumes</td>
<td>Displays the box for each volume that is part of the cloud. Enable this parameter to see the general shape of the cloud. For more information, see Setting Up Cloud Generation Areas (p. 843).</td>
</tr>
</tbody>
</table>
### Generation Parameters

The *Generation* parameters define variables to create clouds through procedural generation.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Display Bounds</strong></td>
<td>Displays the bounds of the entire cloud including all particles. Enable this parameter to help you place clouds.</td>
</tr>
</tbody>
</table>

### Creating a Loop Box

When you add a *Sky Cloud* component to an entity, you also add a *Box Shape* component to function as a loop box. The loop box specifies the bounds in which the cloud can move. When the cloud reaches one side of the loop box, the cloud automatically repositions to the other end so that its movement loops.

**To set up the loop box**

1. In the *Entity Outliner*, select the entity with the *Sky Cloud* component.
2. In the *Entity Inspector*, modify the *Box Shape (p. 832)* component. Set the *Dimensions* values to your preferred loop box size.
For example, \( X = 500.00 \) m, \( Y = 500.00 \) m, \( Z = 200.00 \) m.

3. To see the cloud's movement within the loop box, select AutoMove in the Sky Cloud component properties.
Setting Fade Distance

The **Fade Distance** determines where the cloud begins to fade out as it reaches the edge of the loop box. This prevents the cloud from popping in and out at the edges of the box.

With smaller values, the cloud fades out close to the edge of the loop box. With larger values, the cloud begins to fade closer to the middle of the loop box.

**Example**

The following images show two clouds.

Cloud (1) is near the edge of the loop box but appears solid because of a small fade value. Cloud (2) is equally near the edge but appears faded because of a larger fade value.
To set the fade distance

1. In the **Entity Outliner**, select the entity with the **Sky Cloud** component.
2. In the **Entity Inspector**, in the **Sky Cloud** component properties, adjust the **FadeDistance** slider or enter a value in the **FadeDistance** box.

   **Note**
   This value, measured in meters, should not exceed half of the length of the loop box in the direction that the cloud is moving.

### Changing the Cloud Display

The **Sky Cloud** component's **Display** parameters change how the cloud is displayed in the viewport. You can use these parameters to visualize aspects of the cloud.
When **Display Spheres** is selected, the viewport displays a sphere for each cloud particle.

**To display the cloud as spheres**

1. In the **Entity Outliner**, select the entity with the **Sky Cloud** component.
2. In the **Entity Inspector**, in the **Sky Cloud** component properties, select **Display Spheres**.
Example

When **Display Volumes** is selected, the viewport displays the boxes that make up each part of the cloud volume. For more information, see Setting Up Cloud Generation Areas (p. 843).

To display the cloud as volumes

1. In the **Entity Outliner**, select the entity with the **Sky Cloud** component.
2. In the **Entity Inspector**, in the **Sky Cloud** component properties, select **Display Volumes**.
Example

When **Display Bounds** is selected, the viewport displays a box that includes all areas of the cloud.

To display the cloud's bounds

1. In the **Entity Outliner**, select the entity with the **Sky Cloud** component.
2. In the **Entity Inspector**, in the **Sky Cloud** component properties, select **Display Bounds**.
Example

Setting Up Cloud Generation Areas

You can set up cloud generation areas to customize the shape of your cloud. To define these areas, add **Box Shape** components to child entities under the main parent entity.
To define cloud generation areas

1. Create an entity (p. 463) with a descriptive name, such as **CloudGenerator**.
   
   This will be the parent entity for your cloud generation areas.

2. Add the **Sky Cloud** and **Box Shape** components to the entity.

3. In the **Entity Inspector**, for the **Sky Cloud** component, clear the **Fill by Loopbox** check box.
   
   Clearing this option makes the clouds render inside the child entities rather than in the current entity’s box shape.
4. Create another entity with a descriptive name, such as CloudVolume1, and then add the Box Shape component.

5. Modify the dimensions and position of the Box Shape component to accommodate a cloud that you want to create.

6. Repeat the previous two steps, adding new entities until you achieve the configuration that you want for your clouds.
7. To parent all of the entities to the CloudGenerator entity, select all of the entities and drag it to the parent entity.

8. Select the parent entity and in the Entity Inspector, for the Sky Cloud component, click Generate.
Sky Highlight

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Sky Highlight component creates large bursts of light in the sky. You can use this component with the Lightning (p. 657) component to simulate the sky lighting up when a bolt of lightning flashes.

To enable the Sky Highlight component, you must enable the Lightning Arc gem. For more information, see Lightning Arc Gem (p. 1152).

Note
You can add multiple Sky Highlight components to your entities in a level, but the level uses only one of the Sky Highlight components. The last Sky Highlight component to activate takes priority and Lumberyard renders that component.

Example
The following is a sky highlight effect in the distance.
Contents

- Sky Highlight Component Properties (p. 848)
- EBus Request Bus Interface (p. 849)
  - Enable (p. 849)
  - Disable (p. 849)
  - Toggle (p. 850)
  - IsEnabled (p. 850)
  - SetColor (p. 850)
  - GetColor (p. 851)
  - SetColorMultiplier (p. 851)
  - GetColorMultiplier (p. 851)
  - SetVerticalOffset (p. 851)
  - GetVerticalOffset (p. 852)
  - SetSize (p. 852)
  - GetSize (p. 852)
  - Notification Bus Example Script (p. 852)

Sky Highlight Component Properties

Enabled

Sky highlight effect is rendered when the entity activates.

Default value: True
Color

Color of the sky highlight effect. The default color is a pale blue.

Default value: 204, 204, 255

Color Multiplier

Multiplier to apply to the color. You can use this parameter to adjust the color intensity at run time.

Default value: 1

Valid values: 0 to 100

Vertical Offset

Offsets the height of the sky highlight render position. Specify a value to render the sky highlight above or below the entity's transform.

Default value: 0

Valid values: 0 to 100

Size

Size of the sky highlight effect.

Default value: 10

Valid values: 0 to 100

EBus Request Bus Interface

You can use this EBus to communicate to an entity with a Sky Highlight component attached. The EBus is available at game run time and editing and can be accessed from C++, Lua, and the Script Canvas editor.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

Enable

Enables the sky highlight effect.

Parameters

None

Return

None

Scriptable

Yes

Disable

Disables the sky highlight effect.
Parameters

None

Return

None

Scriptable

Yes

Toggle

Toggles whether the sky highlight effect is enabled.

Parameters

None

Return

None

Scriptable

Yes

IsEnabled

Returns whether the sky highlight effect is enabled.

Parameters

None

Return

Type: Boolean

Scriptable

Yes

SetColor

Sets the color of the sky highlight.

The Color Multiplier parameter uses this value to calculate the final sky highlight color.

Parameters

None

Return

None

Scriptable

Yes
**GetColor**

Returns the color of the sky highlight.

**Note**
This value is not affected by the **Color Multiplier** parameter.

**Parameters**
None

**Return**
None

**Scriptable**
Yes

**SetColorMultiplier**

Sets the color multiplier of the sky highlight.

The **Color** parameter uses this value to calculate the final sky highlight color.

**Parameters**
None

**Return**
None

**Scriptable**
Yes

**GetColorMultiplier**

Returns the color multiplier of the sky highlight.

**Parameters**
None

**Return**
None

**Scriptable**
Yes

**SetVerticalOffset**

Sets how far to offset the sky highlight effect from the entity's transform, on the global z-axis.

**Parameters**
Type: Float
**GetVerticalOffset**

Returns the vertical offset of the sky highlight.

**Parameters**

None

**Return**

Type: Float

**SetSize**

Sets the size of the sky highlight.

**Parameters**

Type: Float

**Return**

None

**GetSize**

Returns the size of the sky highlight.

**Parameters**

None

**Return**

Type: Float

**Notification Bus Example Script**

```function example:OnActivate()```
Snow

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Snow component to create snowfall effects and create patches of snow and ice on the terrain. You can add multiple Snow components to your entities in a level, but the level uses only one Snow component. The last Snow component to activate takes priority and Lumberyard renders that component.

To enable the Snow component, you must enable the Snow gem. For more information, see Snow Gem (p. 1203).

Note
If your level has a Snow and Rain (p. 768) component, the component that activates last takes priority; you cannot enable both Snow and Rain components at the same time.

Contents

• Turning Snow On or Off (p. 854)
• Editing Snow Effects (p. 854)
• Snow Component Properties (p. 854)

Version 1.28
853
• EBus Request Bus Interface (p. 856)
  • Example Snow Component Request Bus (p. 858)

Turning Snow On or Off

You can toggle snow on or off in Lumberyard Editor by enabling or disabling AI/Physics, or by running or stopping the game.

To toggle snow effects on or off in Lumberyard Editor

• Do one of the following:
  • On the bottom right of the viewport, click AI/Physics.
  • Choose Game, Enable Physics/AI.
  • Press Ctrl+P.
  • Press Ctrl+G to enter gameplay mode. Press Esc to stop the game.

Editing Snow Effects

You can edit snow effects only when the AI/Physics toggle is disabled.

To edit snow effects

1. Disable AI/Physics.
2. Make changes to the Snow component properties.
3. Enable AI/Physics. After you reenable AI/Physics, your changes appear in the viewport.

Snow Component Properties

The Snow component has the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Specifies whether snow effects are enabled.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>True</code></td>
</tr>
<tr>
<td>Radius</td>
<td>Radius of snow on the surface.</td>
</tr>
<tr>
<td></td>
<td>Default: 50</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1600</td>
</tr>
<tr>
<td>Snow Amount</td>
<td>Amount of snow on the surface.</td>
</tr>
<tr>
<td></td>
<td>Default: 10</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Frost Amount</td>
<td>Amount of frost on the surface.</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Surface Freezing</td>
<td>Intensity of the surface freezing effect.</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Snowflake Count</td>
<td>Number of generated snowflakes.</td>
</tr>
<tr>
<td></td>
<td>Default: 100</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 10000</td>
</tr>
<tr>
<td>Snowflake Size</td>
<td>Size of the snowflakes.</td>
</tr>
<tr>
<td></td>
<td>Default: 2.5</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Brightness</td>
<td>Brightness of the snowflakes.</td>
</tr>
<tr>
<td></td>
<td>Default: 3</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Gravity Scale</td>
<td>Scale applied to the gravity that affects the snowflakes.</td>
</tr>
<tr>
<td></td>
<td>Default: 0.1</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
<tr>
<td>Wind Scale</td>
<td>Scale applied to the wind that affects the snowflakes.</td>
</tr>
<tr>
<td></td>
<td>Default: 0.1</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 100</td>
</tr>
</tbody>
</table>
### Property | Description
---|---
Turbulence | Turbulence that is applied to the snowflakes.
  - Default: 0.1
  - Valid values: 0 to 100
Turbulence Frequency | Turbulence frequency that is applied to the snowflakes.
  - Default: 0.1
  - Valid values: 0 to 100

### EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game. You can use this EBus to communicate to an entity with a **Snow** component attached. The EBus is available at game time and editing, and can be accessed from C++, Lua, and the **Script Canvas** editor.

When set, the individual setters on this bus update the snow simulation immediately; this can cause some performance implications if you want to change multiple parameters. To avoid this, you can set multiple parameters at once with **GetSnowOptions** (p. 858) and **SetSnowOptions** (p. 858).

For more information, see Working with the Event Bus (EBus) system (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Enables snow effects.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables snow effects.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Toggle</td>
<td>Toggles whether snow effects are enabled.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>IsEnabled</td>
<td>Returns true if the snow effects are enabled.</td>
<td>None</td>
<td>Type: Boolean</td>
<td>Yes</td>
</tr>
<tr>
<td>SetRadius</td>
<td>Sets the radius of the snow surface.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetRadius</td>
<td>Returns the radius of the snow surface.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSnowAmount</td>
<td>Sets the amount of snow on the surface.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSnowAmount</td>
<td>Returns the amount of snow on the surface.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------</td>
<td>----------------</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>SetFrostAmount</td>
<td>Sets the amount of frost on the surface.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetFrostAmount</td>
<td>Returns the amount of frost on the surface.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSurfaceFreezing</td>
<td>Sets the intensity of the freezing effect on the surface.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSurfaceFreezing</td>
<td>Returns the intensity of the freezing effect on the surface.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSnowflakeCount</td>
<td>Sets the number of snowflakes that are created.</td>
<td>Type: AZ::u32</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSnowflakeCount</td>
<td>Returns the number of snowflakes that are created.</td>
<td>None</td>
<td>Type: AZ::u32</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSnowflakeSize</td>
<td>Sets the size of snowflakes that are created.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSnowflakeSize</td>
<td>Returns the size of snowflakes that are created.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSnowflakeBrightness</td>
<td>Sets the brightness of the snowflakes.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSnowflakeBrightness</td>
<td>Returns the brightness of the snowflakes.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSnowflakeGravityScale</td>
<td>Sets the scale applied to the gravity that affects the snowflakes.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetSnowflakeGravityScale</td>
<td>Returns the scale applied to the gravity that affects the snowflakes.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetSnowFallWindScale</td>
<td>Sets the scale applied to the wind that affects the snowflakes.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------</td>
<td>------------</td>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td><code>GetSnowFallWindScale</code></td>
<td>Returns the scale applied to the wind that affects the snowflakes.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SetSnowFallTurbulence</code></td>
<td>Sets the turbulence that is applied to the snowflakes.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><code>GetSnowFallTurbulence</code></td>
<td>Returns the turbulence that is applied to the snowflakes.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SetSnowFallTurbulenceFreq</code></td>
<td>Sets the turbulence frequency that is applied to the snowflakes.</td>
<td>Type: Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><code>GetSnowFallTurbulenceFreq</code></td>
<td>Returns the turbulence frequency that is applied to the snowflakes.</td>
<td>None</td>
<td>Type: Float</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SetSnowOptions</code></td>
<td>Sets all options for the Snow component.</td>
<td>SnowOptions</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><code>GetSnowOptions</code></td>
<td>Returns a data structure that contains all options for the Snow component.</td>
<td>None</td>
<td>SnowOptions</td>
<td>Yes</td>
</tr>
<tr>
<td><code>UpdateSnow</code></td>
<td>Updates Lumberyard to use the Snow component as the base for snow effects. The last Snow component that is called last takes priority; Lumberyard uses that component.</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example Snow Component Request Bus**

```csharp
function example:OnActivate()
    SnowComponentRequestBus.Event.SetFrostAmount(self.entityId, 10)
```
Use the Spawner component to spawn a design-time or runtime dynamic slice (*.dynamicslice) at an entity's location with an optional offset.

Contents

- Spawner Component Properties (p. 859)
- EBus Request Bus Interface (p. 860)
  - Spawn (p. 860)
  - SpawnRelative (p. 860)
  - SpawnAbsolute (p. 860)
  - SpawnSlice (p. 861)
  - SpawnSliceRelative (p. 861)
  - SpawnSliceAbsolute (p. 861)
  - DestroySpawnedSlice (p. 862)
  - DestroyAllSpawnedSlices (p. 862)
  - GetCurrentSpawnedSlices (p. 862)
  - HasAnyCurrentlySpawnedSlices (p. 863)
  - GetCurrentEntitiesFromSpawnedSlice (p. 863)
  - GetAllCurrentlySpawnedEntities (p. 863)
- EBus Notification Bus Interface (p. 864)
  - OnSpawned (p. 864)
  - OnSpawnBegin (p. 864)
  - OnSpawnEnd (p. 864)
  - OnEntitySpawned (p. 865)
  - OnEntitiesSpawned (p. 865)
  - OnSpawnedSliceDestroyed (p. 865)

Spawner Component Properties

The Spawner component has the following properties:

**Dynamic slice**

The slice to spawn.

**Spawn on activate**

If selected, the component spawns the selected slice, upon activation.

Default value: False

**Destroy on deactivate**

If selected, the component destroys any slices that it spawned, upon deactivation.
Default value: False

**EBus Request Bus Interface**

Use the following request functions with the **Spawner** component EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see *Working with the Event Bus (EBus) system* (p. 1851).

A **SliceInstantiationTicket** is the ID for the slice that spawns. You can use this ID with the **Spawner** component to interact with the entities that came from the spawned slice. For example, you can use the **DestroySpawnedSlice** (p. 862) function to destroy all entities from that spawned slice.

**Spawn**

Spawns the selected slice at the entity's location.

**Parameters**

- None

**Return**

- The ID of the spawning slice.

  Type: **SliceInstantiationTicket**

**Scriptable**

- Yes

**SpawnRelative**

Spawns the selected slice at the entity's location with the specified relative offset.

**Parameters**

- **offset** – The coordinates to offset the slice when it spawns.

  Type: Transform

**Return**

- The ID of the spawning slice.

  Type: **SliceInstantiationTicket**

**Scriptable**

- Yes

**SpawnAbsolute**

Spawns the provided slice at the specified world transform.

**Parameters**

- **world** – The world coordinates.
Type: Transform

Return
The ID of the spawning slice.
Type: SliceInstantiationTicket

Scriptable
Yes

SpawnSlice
Spawns the selected slice at the entity's location.

Parameters
slice – The dynamic slice asset.

Return
The ID of the spawning slice.
Type: SliceInstantiationTicket

Scriptable
No

SpawnSliceRelative
Spawns the selected slice at the entity's location with the specified relative offset.

Parameters
slice – The dynamic slice asset.
offset – The coordinates to offset the slice when it spawns.

Type: Transform

Return
The ID of the spawning slice.
Type: SliceInstantiationTicket

Scriptable
No

SpawnSliceAbsolute
Spawns the selected slice at the specified world transform.

Parameters
slice – The dynamic slice asset.
world – The world coordinates.
Spawner

Type: Transform

Return

The ID of the spawning slice.

Type: SliceInstantiationTicket

Scriptable

No

DestroySpawnedSlice

Destroys all entities from the slice that spawned. If the slice is not finished spawning, the slice is canceled. The Spawner component can destroy only slices that it spawned.

Parameters

ticket – The ID of the spawned slice.

Type: SliceInstantiationTicket

Return

None

Scriptable

Yes

DestroyAllSpawnedSlices

Destroys all entities that the Spawner component spawned. Slices that are not finished spawning are canceled.

Parameters

None

Return

None

Scriptable

Yes

GetCurrentlySpawnedSlices

Returns the IDs for spawned slices that are not yet destroyed. A slice is considered destroyed when all its entities are destroyed. The function also includes slices that have not yet finished spawning. This function returns only slices spawned by this Spawner component.

Parameters

None

Return

Type: Vector of SliceInstantiationTickets.
Scriptable

Yes

HasAnyCurrentlySpawnedSlices

Returns whether this Spawner component has spawned any slices, including any slices that are not yet destroyed. A slice is considered destroyed when all its entities are destroyed. Returns `true` if any slices have not yet finished spawning.

Parameters

None

Return

Type: Boolean

Scriptable

Yes

GetCurrentEntitiesFromSpawnedSlice

Returns the IDs of current entities from a slice that spawned. Note that spawning is not instant; if a slice is still spawning, then the entities are not returned. If an entity has been destroyed since it was spawned, its ID is not returned. This function can query only slices spawned by this Spawner component.

Parameters

ticket – The ID of the spawned slice.

Type: SliceInstantiationTicket

Return

Type: Vector of entity IDs.

Scriptable

Yes

GetAllCurrentlySpawnedEntities

Returns the IDs of all existing entities spawned by this Spawner component. Spawning is not instant; if a slice is still spawning, then entities are not returned. If an entity has been destroyed since it was spawned, its ID is not returned.

Parameters

None

Return

Type: Vector of entity IDs.

Scriptable

Yes
EBus Notification Bus Interface

Use the following EBus notification functions with the Spawner component to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

OnSpawned

Notifies that a slice finished spawning.

Parameters

spawnedEntities – Entities that the Spawner component spawned.

Type: Vector of entity IDs.

Return

None

Scriptable

Yes

OnSpawnBegin

Notifies that the slice is beginning to spawn its entities.

Parameters

ticket – The ID of the spawned slice.

Type: SliceInstantiationTicket

Return

None

Scriptable

Yes

OnSpawnEnd

Notifies that the slice finished spawning its entities.

Parameters

ticket – The ID of the spawned slice.

Type: SliceInstantiationTicket

Return

None

Scriptable

Yes
OnEntitySpawned
Notifies that an entity spawned.

Parameters

  ticket – The ID of the spawned slice.
  Type: SliceInstantiationTicket

  spawnedEntity – The entity ID.

Return

  None

Scriptable

  Yes

OnEntitiesSpawned
Notifies that the entities spawned from a slice and sends a list of the entity IDs.

Parameters

  ticket – The ID of the spawned slice.
  Type: SliceInstantiationTicket

  spawnedEntities – The IDs of the entities.
  Type: A vector of entity IDs.

Return

  None

Scriptable

  No

OnSpawnedSliceDestroyed
Notifies when a slice that spawned is destroyed. This occurs when all entities from a spawn are destroyed or when the slice fails to spawn.

Parameters

  ticket – The ID of the spawned slice.
  Type: SliceInstantiationTicket

Return

  None

Scriptable

  Yes
This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create lines and curves in Lumberyard Editor by using the Spline component,

A spline is a curve that connects two or more specific points. This is useful if you want to animate entities along a spline, such as a road or river.

Topics
• Spline Properties (p. 866)
• Working with Spline Components (p. 867)
• EBus Request Bus Interface (p. 873)

Spline Properties

The Spline component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
</tbody>
</table>
| Spline Type | Interpolation type to use between vertices. The component supports the following spline types:  
  • Linear
  • Bézier
  • Catmull-Rom |
| Closed      | Specifies whether a spline is a closed or open loop. |
| Granularity | This property is available only for Bézier and Catmull-Rom spline types.  
  The number of steps for each segment that are in the spline. A step is one of the many short lines that are arranged to approximate the |
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shape of the curve. A segment is the part of the spline between two vertices.</td>
</tr>
<tr>
<td><strong>Knot Parameterization</strong></td>
<td>This property is available only for the Catmull-Rom spline type.</td>
</tr>
<tr>
<td></td>
<td>Adjusts the curvature and smoothness of the spline. Specify a value to support parameterization.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 creates a standard Catmull-Rom spline. A value of 1 creates a chordal Catmull-Rom spline.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Knot Parameterization examples (p. 872).</td>
</tr>
<tr>
<td><strong>Edit</strong></td>
<td>Choose Edit and the component is locked for editing. For more information, see Editing Components in the Viewport (p. 483).</td>
</tr>
</tbody>
</table>

By default, a Spline component has four vertices that are spaced evenly in a line. The vertex positions are stored in the local space of the entity.

![Spline component with vertices and manipulators](image)

**Working with Spline Components**

You can move the vertices in the local space of the entity along each axis. Each vertex has a planar, linear, and surface manipulator. Combined, these are called the translation manipulator. You can use the translation manipulator to move the vertices.

**To change the vertex positions for a spline**

1. Do one of the following:
   - In the viewport, double-click the spline.
   - In the Entity Inspector, choose Edit. For more information, see Editing Components in the Viewport (p. 483).
2. Do one of the following:
Spline

- Select a vertex on the component and drag either the blue, green, or red arrow (linear manipulator) or the hollow blue, green, or red square (planar manipulator) to a position.
- Select a vertex on the component and drag the yellow point (surface manipulator) to move the vertex.

**Note**
The manipulators follow the grid snap setting that you can specify in Lumberyard Editor toolbar. For more information, see Using the Top Toolbar (p. 200).

You can also select multiple vertices.

**To select multiple vertices on a spline**
- Hold Ctrl and select the vertices. Selected vertices appear yellow.
Currently, you can't click and draw a box around the vertices to select them.

**Note**
Currently, you can't click and draw a box around the vertices to select them.

You can also snap a vertex to another position on the terrain.

**To snap a vertex to another position**
1. Select a vertex.
2. Hold Shift+Ctrl and click another position on the terrain. The vertex snaps to that position.

You can also add vertices to the spline.

**To add a vertex to a spline**
1. Pause over a line and hold Ctrl. A preview appears where you can add the vertex.
2. Click to add the vertex to the spline.
You can also delete vertices from the spline.

**To delete a vertex from the spline**

1. Hold Alt and pause on a vertex. The vertex appears gray.

2. Click the vertex to delete it. You can also select a vertex and press Delete.

**Note**

Linear and Bézier splines must have a minimum of two vertices. Catmull-Rom splines must have a minimum of four vertices.
If you change the spline type, the component is updated to represent the new interpolation method.

**Example Bézier**
Note
The first and last vertices of a Catmull-Rom spline are only control points. They don't form part of the curve. A well-formed Catmull-Rom spline must have a minimum of four vertices. The spline doesn't render if there are fewer than four vertices.

The following examples show how Catmull-Rom splines change when you adjust the Knot Parameterization property.

Example Knot Parameterization = 0
Example Knot Parameterization =1

Example Linear and Bézier Splines

The following examples are linear and Bézier splines with the same vertices but with open and closed loops.

EBus Request Bus Interface

Use the following request functions with the SplineComponentRequestBus EBus interface to communicate with other components of your game. The Spline component also uses VertexContainer functions. For more information, see Vertex Containers (p. 904).

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).
<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSpline</td>
<td>Returns a constant pointer to the underlying spline type. You can use this function to query the spline against raycasts and positions. You can also request information, such as the length of the spline, its position, normal, and tangent at various points along the spline.</td>
<td>None</td>
<td>AZ::ConstSplinePtr</td>
<td>Yes</td>
</tr>
<tr>
<td>ChangeSplineType</td>
<td>Changes the spline type. The AZ::u64 value refers to the RTTI hash of the underlying spline type. For example, AZ::LinearSpline::RTTI_Type().GetHash().</td>
<td>AZ::u64</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>SetClosed</td>
<td>Specifies whether the spline forms a closed or open loop.</td>
<td>Boolean</td>
<td>None</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

The following script uses the request bus interface.

```lua
local spline = SplineComponentRequestBus.Event.GetSpline(self.entityId);

-- GetNearestAddressPosition
local posSplineQueryResult = spline:GetNearestAddressPosition(Vector3(0, 0, 0));

posSplineQueryResult.splineAddress
posSplineQueryResult.splineAddress.segmentIndex
posSplineQueryResult.splineAddress.segmentFraction
posSplineQueryResult.distanceSq

-- GetNearestAddressRay
local raySplineQueryResult = spline:GetNearestAddressRay(Vector3(10, -1, 0), Vector3(0, -1, 0));

raySplineQueryResult.splineAddress.segmentIndex
raySplineQueryResult.splineAddress.segmentFraction
raySplineQueryResult.distanceSq
raySplineQueryResult.rayDistance

local splineAddressFromDistance = spline:GetAddressByDistance(3.0);
```
local splineAddressFromFraction = spline:GetAddressByFraction(0.75);

-- Test SplineAddress Constructor (1 param)
local splineAddress_1 = SplineAddress(2);

-- Test SplineAddress Constructor (2 params)
local splineAddress_2 = SplineAddress(1, 0);

spline:GetPosition(splineAddress_1)
spline:GetNormal(splineAddress_1)
spline:GetTangent(splineAddress_1)
spline:GetLength(splineAddress_1)
spline:GetSplineLength()
spline:GetSegmentLength(1)
spline:GetSegmentCount()
spline:GetSegmentGranularity()

local aabb = spline:GetAabb(Transform:CreateIdentity());

local closed = spline:IsClosed();
local firstVertex = spline.vertexContainer[1];
local lastVertex = spline.vertexContainer[spline.vertexContainer:Size()];

-- Attention: You can use the bus interface directly, but here (at the moment at least) indexing will start from 0, not 1 as is the norm in Lua. Instead, you should use the functions directly on the VertexContainer, listed below
SplineComponentRequestBus.Event.AddVertex(self.entityId, lastVertex + Vector3(1, 2, 3));
SplineComponentRequestBus.Event.UpdateVertex(self.entityId, 0, firstVertex + Vector3(1, 2, 3));
SplineComponentRequestBus.Event.InsertVertex(self.entityId, spline.vertexContainer:Size() - 1, lastVertex + Vector3(1, 2, 3));
SplineComponentRequestBus.Event.ClearVertices(self.entityId);
SplineComponentRequestBus.Event.RemoveVertex(self.entityId, spline.vertexContainer:Size() - 1);

-- Prefer these functions - indexing will start from 1
spline.vertexContainer:AddVertex(lastVertex + Vector3(1, 2, 3));
lastVertex + Vector3(1, 2, 3));
spline.vertexContainer:UpdateVertex(1, firstVertex + Vector3(1, 2, 3));
spline.vertexContainer:InsertVertex(spline.vertexContainer:Size(), lastVertex + Vector3(1, 2, 3));
spline.vertexContainer:ClearVertices();
spline.vertexContainer:RemoveVertex(spline.vertexContainer:Size());

Tag

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Tag component to apply one or more labels, or tags, to an entity such as burning or player. You can use these tags to find or filter entities with particular traits. For example, you can set a weapon to inflict double damage to entities tagged as burning.

EBuses – Request Bus Interface: TagGlobalRequestBus

Use the following request function with the TagGlobalRequestBus EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).
RequestTaggedEntities

Handlers respond if they have the tag (listening on the tag's channel). Use AZ::EBusAggregateResults to handle more than the first responder.

Parameters
None

Return
const AZ::EntityId

Scriptable
Yes

EBuses – Request Bus Interface: TagRequestBus

Use the following request functions with the TagRequestBus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

HasTag

Returns true if the entity has the tag.

Parameters
const Tag&

Return
bool

Scriptable
Yes

AddTag

Adds the tag to the entity if it didn't already have it.

Parameters
const Tag&

Return
None

Scriptable
Yes

AddTags

Add a list of tags to the entity if it didn't already have them.
**Tag**

**Parameters**

const Tags&

**Return**

None

**Scriptable**

No

**RemoveTag**

Removes a tag from the entity if it had it.

**Parameters**

const Tag&

**Return**

None

**Scriptable**

Yes

**RemoveTags**

Removes a list of tags from the entity if it had them.

**Parameters**

const Tags&

**Return**

None

**Scriptable**

No

**GetTags**

Gets the list of tags on the entity.

**Parameters**

None

**Return**

const Tags&

**Scriptable**

No

The following is an example of script using the **Request Bus Interface**.

```cpp
local enemies = TagGlobalRequestBus.Event.RequestTaggedEntities(Crc32("Enemy"));
```
local burning = TagComponentRequestBus.Event.HasTag(self.entityId, Crc32("Burning"));

**EBus – Notification Bus Interface:**
**TagComponentNotificationsBus**

Use the following request functions with the **TagComponentNotificationsBus** notification bus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see *Working with the Event Bus (EBus) system* (p. 1851).

**OnTagAdded**
Notifies listeners about tags being added.

**Parameters**
- const Tag& – Indicates the tag was added

**Return**
None

**Scriptable**
Yes

**OnTagRemoved**
Notifies listeners about tags being removed.

**Parameters**
- const Tag& – Indicates the tag was removed

**Return**
None

**Scriptable**
Yes

**Transform**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Transform** component controls the translation, rotation, and scale information of an entity in the 3D world. When you create an entity in Lumberyard Editor, the **Transform** component is automatically added. The translation is the coordinate location (x, y, and z axes) of the entity. The rotation is the degree in which the entity is rotated around its center. The scale is the dimension of the entity in comparison to its original size.

*World space* refers to the entity’s absolute translation, rotation, and scale in the level. If a child is attached to a parent entity, *local space* refers to the entity’s translation, rotation, and scale relative to its parent entity.
Transform Component Properties

The **Transform** component has the following properties:

**Parent entity**

The entity assigned as the parent. If a parent entity is specified, the **Transform** component follows the parent entity.

**Values**

The **Transform** component has the following values:

**Translate**

The local position (relative to the parent) in meters.

**Rotate**

The local rotation (relative to the parent) in degrees.

**Scale**

The local scale.

**Parent activation**

Configures transform behavior when the parent entity activates.

**Static**

Entities that can't be moved at run time. Some systems in Lumberyard treat static entities differently than movable entities (for example, the renderer can optimize static entities, making them less resource intensive to draw).

**Network Sync**

The **Transform** component has following network sync options:

**Sync to replicas**

Transform component syncs in a networked game. If you enable this option, you must also add the **Network Binding** (p. 700) component. Otherwise, the entity is not replicated and the interpolation options don't apply.

**Position Interpolation**

The smoothing of position between network updates and interruptions. This is useful if your objects change location and you notice visual jitter or sudden changes in orientation due to network conditions.

**Rotation Interpolation**

The smoothing of rotation between network updates and interruptions. This is useful if your objects rotate and you notice visual jitter or sudden changes in orientation due to network conditions.

**Note**

Scale interpolation is not supported in the **Transform** component.

**EBus Request Bus Interface**

**TransformBus** is the request bus for the **Transform** component. An entity's transform is the translation, rotation, and scale information.
For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

Use the following request functions with the EBus interface to communicate with other components of your game.

**GetLocalTM**

Returns the entity's local transform. Doesn't include the parent entity's transform.

**Parameters**

None

**Return**

Entity's local transform.

**SetLocalTM**

Sets the entity's local transform, relative to its parent entity, and notifies all listeners.

**Parameters**

Entity's local transform.

**Return**

None

**GetWorldTM**

Returns the entity's world transform, including the parent entity's transform.

**Parameters**

None

**Return**

Entity's world transform.

**SetWorldTM**

Sets the world transform and notifies all listeners.

**Parameters**

Entity's world transform.

**Return**

None

**GetLocalAndWorld**

Retrieves the entity's local and world transforms.

**Parameters**

Transform [out] – Local transform, relative to parent entity.
Transform [out] – World transform.

**Return**

None

**SetWorldTranslation**

Sets the entity's world space translation.

**Parameters**

New world space location, in x, y, and z coordinates.

Type: Vector3

**Return**

None

**SetLocalTranslation**

Sets the entity's local space translation, which is relative to its parent entity.

**Parameters**

New local space location, in x, y, and z coordinates.

Type: Vector3

**Return**

None

**GetWorldTranslation**

Gets the entity's world space translation.

**Parameters**

None

**Return**

Entity's world space, in x, y, and z coordinates.

Type: Vector3

**GetLocalTranslation**

Gets the entity's local space translation, which is relative to its parent entity.

**Parameters**

None

**Return**

Entity's local space, in x, y, and z coordinates.

Type: Vector3
**MoveEntity**
Moves the entity within world space.

**Parameters**
- Offset in world space, in x, y, and z coordinates.
  - Type: Vector3

**Return**
None

**SetWorldX**
Sets the entity's translation x-axis coordinate in world space.

**Parameters**
- X-axis coordinate in world space.
  - Type: Float

**Return**
None

**SetWorldY**
Sets the entity's translation y-axis coordinate in world space.

**Parameters**
- Y-axis coordinate in world space.
  - Type: Float

**Return**
None

**SetWorldZ**
Sets the entity's translation z-axis coordinate in world space.

**Parameters**
- Z-axis coordinate in world space.
  - Type: Float

**Return**
None

**GetWorldX**
Gets the entity's translation x-axis coordinate in world space.

**Parameters**
None
Return

- X-axis coordinate in world space.
  Type: Float

GetWorldY

Gets the entity's translation y-axis coordinate in world space.

Parameters
- None

Return

- Y-axis coordinate in world space.
  Type: Float

GetWorldZ

Sets the entity's translation z-axis coordinate in world space.

Parameters
- None

Return

- Z-axis coordinate in world space.
  Type: Float

SetLocalX

Sets the entity's translation x-axis coordinate in local space.

Parameters
- X-axis coordinate in local space.
  Type: Float

Return

- None

SetLocalY

Sets the entity's translation y-axis coordinate in local space.

Parameters
- Y-axis coordinate in local space.
  Type: Float

Return

- None
SetLocalZ
Sets the entity's translation z-axis coordinate in local space.

Parameters
Z-axis coordinate in local space.
Type: Float

Return
None

GetLocalX
Gets the entity's translation x-axis coordinate in local space.

Parameters
None

Return
X-axis coordinate in local space.
Type: Float

GetLocalY
Gets the entity's y-axis coordinate in local space.

Parameters
None

Return
Y-axis coordinate in local space.
Type: Float

GetLocalZ
Gets the entity's z-axis coordinate in local space.

Parameters
None

Return
Z-axis coordinate in local space.
Type: Float

GetWorldRotation
Gets the angles in radians for each principle axis around which the world transform is rotated in the following order: z-axis, y-axis, x-axis.
Parameters

None

Return

The Euler angles in radians, which indicate the degree of rotation around each principle axis.

Type: Vector3

GetWorldRotationQuaternion

Gets the quaternion that represents the world rotation.

Parameters

None

Return

The quaternion that represents the world rotation.

Type: Quaternion

SetLocalRotation

Sets the local rotation around each principle axes in the following order: z-axis, y-axis, x-axis.

Parameters

The Vector3 denoting radian angles of the rotations around each principle axis.

Type: Vector3

Return

None

SetLocalRotationQuaternion

Sets the local rotation matrix using a quaternion.

Parameters

The local rotation matrix.

Type: Quaternion

Return

None

RotateAroundLocalX

Rotates around the local x-axis for a radian angle.

Parameters

The radian angle to rotate around the local x-axis.

Type: Float
**GetLocalRotation**

Gets angles in radian for each principle axis around which the local transform is rotated in the following order: z-axis, y-axis, x-axis.

**Parameters**

None

**Return**

Indicates how much in radian is rotated around each principle axis.

Type: Vector3

**GetLocalRotationQuaternion**

Gets the quaternion representing the local rotation.

**Parameters**

None

**Return**

The quaternion that represents the local rotation.

Type: Quaternion
SetLocalScale
Sets local scale of the transform.

Parameters
Local scale of the transform, in x, y, and z coordinates.
Type: Vector3

Return
None

SetLocalScaleX
Sets local scale of the transform on the x-axis.

Parameters
X-axis coordinate for the local scale.
Type: Float

Return
None

SetLocalScaleY
Sets local scale of the transform on the y-axis.

Parameters
Y-axis coordinate for the local scale.
Type: Float

Return
None

SetLocalScaleZ
Sets local scale of the transform on the z-axis.

Parameters
Z-axis coordinate for the local scale.
Type: Float

Return
None

GetLocalScale
Gets the scale value on each axis in local space.

Parameters
None
**Return**

Scale value for each axis in local space.

Type: Vector3

**GetWorldScale**

Gets the scale value on each axis in world space. Note that the transform is skewed when it is rotated and has a parent transform scale in which the returned world scale from this function is inaccurate.

**Parameters**

None

**Return**

Scale values for each axis in world space.

Type: Vector3

**GetParentId**

Returns the parent entity's ID. If the entity does not have a parent, the entity ID is invalid.

**Parameters**

None

**Return**

EntityID of the parent

Type: Int

**SetParent**

Sets the entity's parent entity and notifies all listeners. The entity's local transform is moved into the parent entity's space to preserve the entity's world transform.

**Parameters**

EntityId – Parent entity ID

Type: Int

**Return**

None

**SetParentRelative**

Sets the entity's parent entity, moves the transform relative to the parent entity, and notifies all listeners. This function uses the world transform as a local transform and moves the transform relative to the parent entity.

**Parameters**

EntityId – Parent entity ID

Type: Int
Return
None

GetChildren
Returns the entity IDs of the entity's immediate children.

Parameters
None

Return
Vector of EntityIds

GetAllDescendants
Returns the entity IDs of all descendants of the entity. The descendants are the entity's children, the children's children, and so on. The entity IDs are ordered breadth first.

Parameters
None

Return
Vector of EntityIds

GetEntityAndAllDescendants
Returns the entity ID of the entity and all its descendants. The descendants are the entity's children, the children's children, and so on. The entity IDs are ordered breadth first, and this entity's ID is the first in the list.

Parameters
None

Return
Vector of EntityIds

IsStaticTransform
Returns whether the transform is static. A static transform doesn't move and doesn't respond to requests to move it.

Parameters
None

Return
Boolean

EBus Notification Bus Interface

TransformNotificationBus is the notification bus for the Transform component. Use the following notification functions with the EBus interface to communicate with other components of your game.
For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**OnTransformChanged**
Signals that the local or world transform of the entity changed.

**Parameters**
- Transform – The new local transform of the entity
- Transform – The new world transform of the entity

**OnParentChanged**
Signals that the parent of the entity changed.

**Parameters**
- EntityId – The entity ID of the previous parent. The entity ID is invalid if there was no previous parent.
- EntityId – The entity ID of the new parent. The entity ID is invalid if there is no new parent.

**OnChildAdded**
Signals that a child was added to the entity.

**Parameters**
- EntityId – The entity ID of the added child

**OnChildRemoved**
Signals that a child was removed from the entity.

**Parameters**
- EntityId – The entity ID of the removed child

**Trigger Area**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Trigger Area component provides generic triggering services by using Shape (p. 818) components as its bounds.

**To add a trigger component**

1. Create a new entity, or add the Trigger Area component to an existing entity.
2. In the Entity Inspector, click Add Component, then select Trigger Area.

   Entity Inspector displays a message, "This component is missing a required component service and has been disabled."
3. Click **Add Required Component**, then click one of the shape components that appears.

   The shape component you select defines the boundaries for the trigger.

4. If you want to change the boundaries of your shape component, simply remove the existing shape component, and then add a different shape component.

   **Note**
   - The **Trigger Area** component does not work with the PhysX system.

### Trigger Area Component Properties

The **Trigger Area** component has the following properties:

**Network Bindable**

Network bindable components are synchronized over the network.

**Bind To Network**

   When selected, synchronizes component across the network.

**Activation**

**Trigger once**

   If selected, the trigger deactivates after the first trigger event.

**Activated by**

   Select whether trigger is activated by **All entities**, which allows any entity to trigger the area, or by **Specific Entities**, which allows you to select specific entities.

**Tag filters**

**RequiredTags**

   A list of tags that are required for an entity to trigger this area.

**ExcludedTags**

   A list of tags that exclude an entity from triggering this area.

### EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Working with the Event Bus (EBus) system](p. 1851).

**AddRequiredTag**

   Adds a required tag to the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

**Parameters**

   ```cpp
   const Tag&
   ```
requiredTag – Tag to be added
Return
None
Scriptable
Yes

RemoveRequiredTag
Removes a required tag from the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters
const Tag&
    requiredTag – Tag to be removed
Return
None
Scriptable
Yes

AddExcludedTag
Adds an excluded tag to the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters
const Tag&
    excludedTag – Tag to be added
Return
None
Scriptable
Yes

RemoveExcludedTag
Removes an excluded tag from the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters
const Tag& excludedTag – Tag to be removed
Return
None
Scriptable
Yes
EBus Notification Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

The Trigger component sends notifications to:

- Entities listening on the TriggerAreaNotificationBus for the entity with the trigger on it.
- Entities listening on the TriggerAreaEntityNotificationBus for the entity that enters or exits the trigger.

TriggerAreaNotificationBus

This bus allows the game to listen for events associated with a particular trigger. Notifies of all the entities that enter and exit this trigger.

OnTriggerAreaEntered

Notifies listeners when enteringEntityId enters this trigger.

Parameters

  enteringEntityId – ID of entity that has entered this trigger

Return

  None

Scriptable

  Yes

OnTriggerAreaExited

Notifies listeners when enteringEntityId exits this trigger.

Parameters

  enteringEntityId – ID of entity that has exited this trigger

Return

  None

Scriptable

  Yes

TriggerAreaEntityNotificationBus

This bus allows the game to listen for trigger-related events associated with a particular entity. Notifies every time the player enters or exits any trigger.

OnEntityEnteredTriggerArea

Sent when the entity enters triggerId.

Parameters

  triggerId – ID of entity that the trigger is on
Return

None

Scriptable

Yes

**OnEntityExitedTriggerArea**

Sent when the entity exits triggerID.

**Parameters**

- `triggerId` - *ID of entity that the trigger is on*

**Return**

None

Scriptable

Yes

The following is an example of script using the EBus interface.

```lua
local triggerexample = {
    Properties = {
    }
}

function triggerexample:OnActivate()
    self.triggerHandler = TriggerAreaEntityNotificationBus.Connect(self, self.entityId)
end

function triggerexample:OnDeactivate()
    self.triggerHandler:Disconnect()
end

function triggerexample:OnEntityEnteredTriggerArea(entityId)
    Debug.Log("------- TRIGGERED.")
end

return triggerexample
```

**Tube Shape**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Create tube-like volumes along a spline by using the **Tube Shape** component. To create a tube shape, add the **Spline** component to an entity to define its shape. In the **Tube Shape** component, define a radius to control its volume. You can specify a different tube radius at each vertex in the spline.
Note
The Tube Shape component requires the Spline (p. 866) component.

Topics
- Tube Shape Properties (p. 895)
- EBus Request Bus Interface (p. 896)

Example Tube Shape Component

Tube Shape Properties

![Tube Shape Properties](image)

Version 1.28
895
The **Tube Shape** component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>The component always appears in the viewport, even if the entity isn't selected.</td>
</tr>
<tr>
<td>Game View</td>
<td>Renders the shape in game mode.</td>
</tr>
<tr>
<td></td>
<td>To enter game mode, press <strong>Ctrl + G</strong>.</td>
</tr>
<tr>
<td>Shape Color</td>
<td>Specifies the color to render the shape.</td>
</tr>
<tr>
<td>Radius</td>
<td>The radius of the tube.</td>
</tr>
<tr>
<td>Variable Radius</td>
<td>Defines the variable radius at each point along the spline. This value is added to the radius to calculate the final radius of the tube.</td>
</tr>
<tr>
<td>Edit</td>
<td>Choose <strong>Edit</strong>, and the component is locked for editing. For more information, see <strong>Editing Components in the Viewport</strong> (p. 483).</td>
</tr>
</tbody>
</table>

**EBus Request Bus Interface**

You can use the event bus (EBus) interface to communicate with other components in your game. For more information, see **Working with the Event Bus (EBus) system** (p. 1851).

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Return</th>
<th>Scriptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRadius</td>
<td>Returns the radius of the tube.</td>
<td>Void</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>GetTotalRadius</td>
<td>Returns the total interpolated radius of the tube. This is the sum of the radius and the variable radius.</td>
<td>SplineAddress</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>GetVariableRadius</td>
<td>Returns the variable radius along the spline.</td>
<td>Int</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetRadius</td>
<td>Sets the radius of the tube.</td>
<td>Float</td>
<td>Void</td>
<td>Yes</td>
</tr>
<tr>
<td>SetVariableRadius</td>
<td>Sets the variable radius of the tube at a spline point.</td>
<td>Int, Float</td>
<td>Void</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

The following script uses the request bus interface.

```lua
function Script:OnActivate()
    TubeShapeComponentRequestsBus.Event.SetRadius(self.entityId, 5.0);
end
```
UI Canvas Asset Ref

With the **UI Canvas Asset Ref** component, you can associate a UI canvas with a component entity in a level.

When you set up a UI canvas asset ref component, you can:

- Select whether to automatically load the UI canvas when the level loads
- Use Script Canvas that is associated with the UI canvas to reference it using the **UI Canvas Asset Ref** Script Canvas nodes.

Use this component in conjunction with the **UI Canvas on Mesh** (p. 897) component if you want to place a UI canvas on a 3D mesh that a player can interact with.

For more information about how to use the **UI Canvas Asset Ref** component, see Placing UI Canvases in the 3D World (p. 2897).

**UI Canvas Asset Ref Component Properties**

The **UI Canvas Asset Ref** component has the following properties:

- **Canvas pathname**
  
  The relative pathname of the UI canvas asset file.

- **Load automatically**
  
  If selected, the canvas is automatically loaded when this component entity is loaded, typically when the level is loaded.

UI Canvas on Mesh

With the **UI Canvas on Mesh** component, you can place a UI canvas on a component entity in the 3D world that a player can interact with via ray casts. Use this component in conjunction with the **UI Canvas Asset Ref** (p. 897) component.

For more information about how to use the **UI Canvas Asset Ref** component, see Placing UI Canvases in the 3D World (p. 2897).

**UI Canvas on Mesh Component Properties**

The UI Canvas on Mesh component has the following properties:

- **Render target override**
  
  For simple cases, you can leave this property blank. The UI canvas specifies a render target, and that render target can be used as a texture name for the material on the 3D mesh.
You can use the **Render target override** property when you want to load two unique instances of the same UI canvas that the user can set to different states. This example case is demonstrated in the **UiIn3DWorld** level in the **Samples Project**.

**Unique Instances of Canvas**

These two entities each load a unique instance of the same canvas. The sphere above casts a shadow on the left entity because it is using the Illum shader.

**UI Canvas Proxy Ref**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With the UI Canvas Proxy Ref component, you can associate a component entity in a level with another component entity that is managing a UI canvas. Use this component in conjunction with the **UI Canvas on Mesh** (p. 897) component if you want to place a UI canvas on a 3D mesh that a player can interact with in several places in the 3D world.
Use of this component is often a special case, as it supports displaying the same UI canvas on multiple entities in the 3D world. The UI Canvas Proxy Ref component allows the component entity that it is on to act as if it had a UI Canvas Asset Ref (p. 897) component but without having to load another copy of the UI canvas. This means that, as the user interacts with one UI canvas on a 3D object, the other 3D object shows the same changes.

To see an example of using the UI Canvas Proxy Ref component, open the UiIn3DWorld level in the Samples Project.

The following picture shows three entities that share the same loaded canvas. The curved plane entity has a UI Canvas Asset Ref component and the egg and the sphere both have UI Canvas Proxy Ref components:

---

UI Canvas Proxy Ref Component Properties

The UI Canvas Proxy Ref component has the following properties:
Canvas Asset Ref entity

Click the picker (hand icon) to select the component entity that you want to mirror. The picked component must have the UI Canvas Asset Ref component on it.

Vegetation Layer Spawner

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Vegetation Layer Spawner component to define areas and rules for procedurally placing dynamic vegetation or other static meshes in your Lumberyard level.

To use the Vegetation Layer Spawner component, you must enable the Vegetation gem.

With the Vegetation Layer Spawner component, you can do the following:

- Create vegetation within a user-defined area at runtime
- Configure layer settings to determine the depth or relative ordering in which to apply vegetation layers
- (Optional) Add modifier and filter components to add variation to placed vegetation
- (Optional) Use selector components to determine which vegetation assets to place at a given location
- Control the preview settings for attached components
- Inherit behavior from a parent Vegetation Layer Blenders component

Add the following required components when using the Vegetation Layer Spawner component:

- A Shape or Vegetation Reference Shape component to define the vegetation's spawn area
- The Vegetation Asset List component to list mesh assets, material assets, and other settings for the vegetation

The following optional components change the Vegetation Layer Spawner component's behavior:

- Vegetation Filter components use various conditions to determine whether to create vegetation
- Vegetation Modifier components change the appearance of generated vegetation
- Vegetation Selector components determine which asset descriptors are selected for creation

For information on how to use the Vegetation Layer Spawner, see Dynamic vegetation (p. 1314).

Vegetation Layer Spawner Properties

The Vegetation Layer Spawner component has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Override Preview Settings</td>
<td>Controls whether user-specified settings or default settings are used to display the preview. Select this option to expose additional related properties.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pin Preview to Shape</td>
<td>Specified shape entity determines the preview position and size. Remove the target entity to expose Preview Position and Preview Size.</td>
</tr>
<tr>
<td>Preview Position</td>
<td>Specifies an offset for the world space location at the center of the box. This box is used for sampling values that the preview window displays.</td>
</tr>
<tr>
<td>Preview Size</td>
<td>Specifies a scale for the world space dimensions of the box that is used for sampling values that the preview window displays.</td>
</tr>
<tr>
<td>Constrain to Shape</td>
<td>When enabled, forces preview widgets to display only content that is in the component's shape instead of the filling the entire image.</td>
</tr>
<tr>
<td>Layer Priority</td>
<td>Defines a high-level priority or depth for vegetation to be sorted and processed relative to other layers.</td>
</tr>
<tr>
<td></td>
<td>All foreground layers are processed and can plant vegetation before all background layers.</td>
</tr>
<tr>
<td>Sub Priority</td>
<td>Defines a priority that is used to sort and apply vegetation areas in the same layer.</td>
</tr>
<tr>
<td></td>
<td>Vegetation layers with higher priorities are applied before those with lower priorities.</td>
</tr>
<tr>
<td>Inherit Behavior</td>
<td>Enables certain vegetation layer components to use the filters and modifiers that are attached to a parent entity in a vegetation layer blender hierarchy.</td>
</tr>
<tr>
<td>Allow Empty Meshes</td>
<td>Enables unspecified or invalid meshes to plant and claim space. Spawners typically plant only visible vegetation instances, which claim space and prevent other vegetation from being planted. By default, unspecified or invalid meshes are skipped.</td>
</tr>
<tr>
<td>Filter Stage</td>
<td>Defines when vegetation filters are processed. Individual vegetation filters can override this setting. For more information, see the text following this table.</td>
</tr>
</tbody>
</table>

For the Filter Stage property, the **PreProcess** option results in faster processing than the **PostProcess** option, which ensures accuracy.

- **PreProcess** – Processes vegetation filters before modifiers. This means that fewer instances are processed for modifiers, which generally process faster. However, modifiers that affect slope, altitude, and so on might affect the criteria that would have filtered out the instance. Running the filters first means that the modified data isn't checked by the filter.
• **PostProcess** – Processes vegetation modifiers before filters. This means that modifiers process every potential instance before filters remove them. Because of the added calculations, this method is slower but ensures correct filtering out of vegetation whose filter criteria has been modified.

**Example**

A Vegetation Altitude Filter is set to filter out vegetation below 32 meters. A Vegetation Position Modifier is set to shift vegetation left by 5 meters. The initial Vegetation Layer Spawner is placed in a box sitting at 32 meters. With the **PreProcess** option (A), the filter is applied first, and then the modifier shifts vegetation left into the canyon. With the **PostProcess** option (B), the modifier shifts vegetation left into the canyon first. The filter is then applied, which removes the vegetation below 32 meters.

![Example Image](image)

**Ebus Request Bus Interface**

The Vegetation Layer Spawner component uses Vegetation::SpawnerRequestBus and Vegetation::AreaInfoBus interfaces.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

Use the following request functions with the EBus interface to communicate with other components of your game.

Use the following EBus requests with the Vegetation::SpawnerRequestBus interface.

<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAreaPriority</td>
<td>Get the priority (in the layer) of the vegetation area. Higher numbers have higher priority.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>SetAreaPriority</td>
<td>Set the priority (in the layer) of the vegetation area.</td>
<td>Float</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetAreaLayer</td>
<td>Get the layer of the vegetation area.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>Request Name</td>
<td>Description</td>
<td>Parameters</td>
<td>Return</td>
<td>Scriptable?</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>SetAreaLayer</td>
<td>Set the layer of the vegetation area.</td>
<td>AreaLayer</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetAreaProductCount</td>
<td>Get the number of currently spawned instances in the vegetation area.</td>
<td>AZ::u32</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>GetInheritBehavior</td>
<td>Get the Inherit Behavior setting, which controls whether shapes, modifiers, and filters of a parent entity affect this area.</td>
<td>Bool</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>SetInheritBehavior</td>
<td>Set the Inherit Behavior setting.</td>
<td>None</td>
<td>Bool</td>
<td>Yes</td>
</tr>
<tr>
<td>GetAllowEmptyMeshes</td>
<td>Get the Allow Empty Meshes setting, which controls whether empty meshes in the Vegetation Asset List are spawned as empty spaces.</td>
<td>Bool</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>SetAllowEmptyMeshes</td>
<td>Set the Allow Empty Meshes setting.</td>
<td>None</td>
<td>Bool</td>
<td>Yes</td>
</tr>
<tr>
<td>GetFilterStage</td>
<td>Get the filter stage. PreProcess means that vegetation filters are applied before vegetation modifiers. PostProcess means that they’re applied afterwards.</td>
<td>FilterStage</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>SetFilterStage</td>
<td>Set the filter stage.</td>
<td>None</td>
<td>FilterStage</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Use the following EBus requests with the Vegetation: :AreaInfoBus interface.
<table>
<thead>
<tr>
<th>Request Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return</th>
<th>Scriptable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPriority</td>
<td>Get the combined layer and area priority value as a single combined global priority.</td>
<td>None</td>
<td>Float</td>
<td>Yes</td>
</tr>
<tr>
<td>GetEncompassingAabb</td>
<td>Get the axis-aligned bounding box for the entire vegetation area.</td>
<td>None</td>
<td>AZ::Aabb</td>
<td>Yes</td>
</tr>
<tr>
<td>GetProductCount</td>
<td>Get the number of currently spawned instances in the vegetation area. Same as GetAreaProductCount.</td>
<td>None</td>
<td>AZ::u32</td>
<td>Yes</td>
</tr>
<tr>
<td>GetChangeIndex</td>
<td>Get an incrementing number that represents the number of times that the spawner area has been refreshed since creation.</td>
<td>None</td>
<td>AZ::u32</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Vertex Containers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

VertexContainer is a concrete type and an interface that is used by several Lumberyard component buses. It is implemented directly by the Polygon Prism Shape (p. 708) component and the Spline (p. 866) component and used indirectly by the Navigation Area (p. 693) component.

### Vertex Container Interface

A vertex container provides the ability to access, modify, add, and remove vertices. A vertex container has several interfaces that build upon one another.

These include the following interfaces:

**Contents**
- FixedVertices Interface (p. 905)
- VariableVertices Interface (p. 905)
- VertexContainerInterface (p. 905)
FixedVertices Interface

The simplest interface is FixedVertices. FixedVertices supports GetVertex and UpdateVertex functions for vertices in a container and a Size function that returns the number of vertices in the container.

Because the number of vertices is fixed, vertices cannot be added or removed. You can implement this interface with an array or AZStd::fixed_vector.

VariableVertices Interface

The VariableVertices interface supports all the functionality of FixedVertices, but also provides AddVertex, InsertVertex, and RemoveVertex functions. VariableVertices also provides a utility function called Empty that checks whether a container has any elements or is empty.

To implement the VariableVertices interface, you can use either an AZStd::vector or a VertexContainer.

VertexContainerInterface

The VertexContainerInterface provides an interface to all functionality provided by the previous two interfaces and the VertexContainer type. For convenience, the VertexContainerInterface also provides SetVertices and ClearVertices functions that can update all vertices or remove all vertices in one operation. The SetVertices function takes a vertices parameter that contains a list of all vertices to be stored.

**Note**

A VertexContainer owns the vertices to which it has access; they are not stored elsewhere (a VertexContainer is not a view).

As mentioned, the VertexContainerInterface is implemented by the Spline component and Polygon Prism component EBuses. The Navigation Area component also uses the interface through its dependence on the Polygon Prism Shape component. Each of these components uses the VertexContainer type internally to manage its vertices.

For more information about the interfaces in the VertexContainerInterface, see the code and code comments in the lumberyard_version\dev\Code\Framework\AzCore\AzCore\Math\VertexContainerInterface.h file.

For more information about the VertexContainer type, see the code and code comments in the lumberyard_version\dev\Code\Framework\AzCore\AzCore\Math\VertexContainer.h file.

**Note**

The VertexContainer can store Vector2 or Vector3 types. The vector type is determined at compile time when the type is created. This is useful for certain components that do not allow points to be modified on the Z (vertical) axis and treat points just in two dimensions. The Polygon Prism component uses the Vector2 type.

EBus Request Bus Interface Example

The following is an example of a Lua script that uses the RequestBus interface.

```lua
local firstVertex = <type_with_vertex_container>.vertexContainer[1];
local lastVertex = <type_with_vertex_container>.vertexContainer[spline.vertexContainer:size()];

...RequestBus.Event.AddVertex(self.entityId, lastVertex + Vector3(1, 2, 3));
...RequestBus.Event.UpdateVertex(self.entityId, 0, firstVertex + Vector3(1, 2, 3));
...RequestBus.Event.InsertVertex(self.entityId, spline.vertexContainer:size() - 1, lastVertex + Vector3(1, 2, 3));
...RequestBus.Event.ClearVertices(self.entityId);
...RequestBus.Event.RemoveVertex(self.entityId, spline.vertexContainer:size() - 1);
```
Video Playback

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Video Playback component to play a video on an entity in your Lumberyard level. For example, you can use a flat or plane entity to simulate a movie screen. You add the video playback component to the entity and specify a video file to display.

You can also play a video on a level loading screen. For more information, see Defining Game and Level Load Screens (p. 2910).

You can use Script Canvas or Lua scripting to trigger the video to play, pause, or stop, depending on player actions.

Prerequisites

To use the Video Playback component, you must do the following:

- Install either FFmpeg or LibAV.
- Enable the Video Playback gem for your game project. See Enabling Gems (p. 1064).
- Rebuild your game project.

**Important**

Two additional steps are required to use FFmpeg in the current Lumberyard release.

1. In the file, \dev\Gems\VideoPlayback\Code\Source\decoder.h, at line 32, insert the following preprocessor directive.

   ```
   #pragma warning(disable : 4996)
   ```

   This preprocessor directive will disable compilation warnings associated with FFmpeg, which would otherwise be treated as errors.

2. Once your project is compiled, you will need to manually copy the .dll files from \3rdParty\FFmpeg\3.2\bin\ to the build output folder \dev\Bin64\vc141_or_vc142

You can also set your video to play in visual stereo (not audio stereo). To test this feature, you must use a virtual reality head mounted display (HMD).

**Note**

Audio isn't currently supported with the Video Playback component, but you can trigger audio playback separately if you want to play audio with your video.

Topics
Setting Up Video Playback

To set up video playback in Lumberyard, you must install either FFmpeg or LibAV.

- If both are installed, Lumberyard uses FFmpeg.
- To use LibAV, remove FFmpeg and see Installing LibAV (p. 907).

**Note**
Certain third-party software might require a license. Consult the terms of service before installing the software.

Installing FFmpeg

To install FFmpeg for Lumberyard, follow these steps.

**To install FFmpeg**

1. Locate a Windows build from the download page on FFmpeg.
2. Download a developer build (sometimes referred to as a "shared" build) that contains bin, include, and lib directories. Unzip the files.
3. Navigate to lumberyard_version/3rdParty and create a directory named FFmpeg.
4. In FFmpeg, create a directory named 3.2.
   **Note**
   You must name the directory 3.2, regardless of the FFmpeg version that you're using.
5. Copy the bin, include, and lib folders to the lumberyard_version/3rdParty/FFmpeg/3.2 directory.
6. Run Lumberyard Setup Assistant and, on the Install optional SDKs page, verify that Lumberyard detects FFmpeg.

Installing LibAV

To install LibAV for Lumberyard, follow these steps.

**To install LibAV**

2. Select the release-lgpl build and download the latest x86_64 version of LibAV. As of this writing, version 11.7.7 is the latest build.
3. Navigate to lumberyard_version/3rdParty and create a directory named libav.
4. In the libav directory, create a directory named 11.7.

   **Note**
   You must name the directory 11.7, regardless of the LibAV version that you’re using.

5. Extract the .7z file to the 11.7 directory.

   **Note**
   To open and extract .7z files, you must use a 7z application, such as 7-Zip.

In the 11.7 directory, you should have the following:

- A directory named usr
- config.log
- md5sum

6. Run Lumberyard Setup Assistant and, on the **Install optional SDKs** page, verify that Lumberyard detects LibAV.

---

### Using the Video Playback Component

After you complete the **Prerequisites (p. 906)**, you can use the **Video Playback** component.

Video playback supports the following container formats:

- .mp4
- .mkv (recommended)
- .webm (recommended)

Video playback supports the following codecs:

- .h264
- .h265
- VP8 (recommended)
- VP9 (recommended)

The basic setup for the **Video Playback** component includes the following:

- Add a **Camera** component
- Add a **Mesh** and **Video Playback** component
- Configure your material

**To use the Video Playback component**

1. If you don’t yet have a camera in your scene, place a **Camera (p. 566)** component where your video playback is to be placed.
You can use the camera to view your video playback. Ensure that the camera is facing the direction where you place your video playback component.

2. Create an entity. For more information, see Creating an Entity (p. 463).

3. Use the Entity Inspector (p. 475) to add a Mesh (p. 684) component to your entity.

4. For the Mesh component, select a Mesh asset. This is the asset that your video renders on. A cube or plane is a good test mesh.

![Mesh component settings](image)

5. Add the Video Playback component to the same entity.

6. In the Video Playback component, for Video, select the video to display.

7. For Texture name, enter dollar sign ($) and a name for your texture. You can enter any name, but it must begin with a $ character to indicate that it's a render target. For example, $videotest is a valid name, but videotest isn't.

![Video Playback component settings](image)

8. For Frame queue ahead count, set the number of frames to buffer.

   We recommend that you use a value from 1 to 3.

   Queueing too many frames to buffer (for example, a value of 100 frames) can use too much memory and cause performance issues.

9. Open the Material Editor (p. 1636).

10. To create a material, click the Add New Item icon. Enter a descriptive name, such as myvideomaterial.
11. Under **Texture Maps**, on the **Diffuse** line, enter the name of your video component’s **Texture name** field. You must include the $ character.
12. Close the **Material Editor** and return to the **Entity Inspector (p. 475)**. In the **Mesh** component, for the **Material override** property, select the material that you created.

![Material Editor Screenshot](image)

You can trigger the video to play at the start of your game using Lua scripting.

**Setting Up Stereo Video Playback**

Before setting up stereo video playback, ensure that you have completed the setup instructions in **Setting Up Video Playback (p. 907)**.

Stereo video playback means that the video presents a slightly different image for each eye, creating a 3D feel. Stereo in this case refers to visual stereo only; audio is not supported in the video playback component. While audio is not supported and must be synced externally, it is possible to play back spatialized audio. To do so, you must use the full, commercial version of Wwise and a 3D spatializer plugin such as Oculus Spatializer or RealSpace 3D.

To set up stereo video playback, your source video file must be laid out for stereo. Lumberyard supports videos in a top-bottom or bottom-top layout. You must have a VR headset to verify that the video is playing in stereo.

To set up stereo video playback, follow the instructions in **Using the Video Playback Component (p. 908)**. The only differences in the setup are the following:

- You must use a source video file with 3D or stereo information
- Set the **Stereo Layout** property initially to **Auto-Detect**. If it fails to auto-detect, manually set it to **Top-Bottom** or **Bottom-Top**.

All supported video files should have their stereo layout written into their metadata. This, however, is not a requirement and may not have been inserted by your encoder. If you would like to inject stereo metadata into your video, see [https://support.google.com/jump/answer/7044297?hl=en](https://support.google.com/jump/answer/7044297?hl=en).

![Video Playback Screenshot](image)

When you enter game mode (using **Ctrl G**), you should see the left eye of your video play. If you do not see this, try changing your **Stereo layout** setting.

To verify that your video is playing in stereo, you must enter VR mode. You can enter VR mode by clicking **VR Preview** at the bottom right corner of the viewport. Then press **Ctrl G** to enter game mode.
If your **VR Preview** button isn’t enabled, or you can’t get into VR preview mode, ensure that your VR headset is working outside of Lumberyard and then restart the Lumberyard editor.

Playing stereo video is resource intensive. Because the video is often close in proximity to the player, it becomes easy to detect inconsistencies and artifacts in the video. To prevent that, use higher resolution videos whenever possible. To conserve resources, do not play more than one or two high resolution stereo videos at a time.

**Lua Bindings for Video Playback**

You can use Lua bindings to interact programmatically with video playback components that you’ve placed in your scene. Lua provides a way to establish complex logic for playing, pausing, and stopping videos.

**Global Functions**

The following functions provide programming interfaces for the video playback systems.

**VideoPlaybackRequestBusSender**

**Parameters**

- **EntityID**

**Return**

Returns the `VideoPlaybackRequestBusSender` object that is connected to the specified entity. For more information, see `VideoPlaybackRequestBusSender Object (p. 912)`.

**VideoPlaybackNotificationBusHandler**

Exposes callbacks to your Lua script that are triggered by events during video playback.

For more information, see `VideoPlaybackNotificationBusHandler Object (p. 913)`.

**Parameters**

- **Table** – The Lua table to which you want to expose the callback functions. Pass `self` to expose the callbacks to the current Lua script.

- **EntityId** –

**Return**

Returns the `VideoPlaybackRequestBusSender` object that is connected to the specified entity. For more information, see `VideoPlaybackRequestBusSender Object (p. 912)`.

**VideoPlaybackRequestBusSender Object**

The `VideoPlaybackRequestBusSender` object contains functions with which you can send requests to the video playback component.

**Bool IsPlaying()**

Returns `true` if the video is playing. If the video is paused or stopped, returns `false`.

**Void Play()**

Plays the video. If no video is selected or the video is already playing, this has no effect.

**Void Pause()**

Pauses the video. If the video is already paused, this has no effect.
Void Stop()

Stops the video and remains on the last frame. When the video plays again, it begins at the first frame of the video. If the video is already stopped, this has no effect. If the video is playing or paused, the video stops.

Void EnableLooping(Bool)

Sets whether this video automatically restarts from the beginning once the end of the video is reached. Pass true to enable looping or false to disable looping. Looping is disabled by default.

Void SetPlaybackSpeed(Float)

Sets how fast the video plays. For example, 1.0 is normal speed, 0.5 is half speed, 2.0 is double speed, and so on.

Caution is advised when setting the video speed. Setting a speed that is too high can result in choppy playback.

VideoPlaybackNotificationBusHandler Object

The VideoPlaybackNotificationBusHandler object exposes callback functions to your Lua script that are triggered by events that happen during video playback.

Void OnPlaybackStarted()

Called when video playback begins.

Void OnPlaybackPaused()

Called when video playback pauses. Not called when video stops.

Void OnPlaybackStopped()

Called when video playback is stopped by the user. If the video reaches the end and is not set to loop, this function is not called.

Void OnPlaybackFinished()

Called when all frames in the video are played. This is not called if the user manually stops video playback. If looping is enabled, this function is called every time the video loops.

Video Playback Bink

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Video Playback Bink component to play a video on an entity in your Lumberyard level. For example, you can use a flat or plane entity to simulate a movie screen. You add the video playback component to the entity and specify a Bink video file to display.

You can also play a video on a level loading screen. For more information, see Defining Game and Level Load Screens (p. 2910).

You can use Script Canvas or Lua scripting to trigger the video to play, pause, or stop, depending on player actions.

Prerequisites

To use the Video Playback Bink component, you must do the following:
- Obtain a Bink license. Contact RAD Game Tools for information on how to license Bink.
- Install the **Video Playback Bink** gem and the Bink libraries into the gem.

**Note**
A Bink license is needed to gain access to this gem. Installation instructions are provided with the gem.

- Enable the **Video Playback Bink** gem for your game project. See Enabling Gems (p. 1064).
- Rebuild your game project.

**Topics**
- Using the Video Playback Bink Component (p. 914)
- Lua Bindings for Video Playback Bink (p. 917)

**Using the Video Playback Bink Component**

After you complete the Prerequisites (p. 913), you can use the **Video Playback Bink** component.

Video playback supports the `.bk2` format and the Bink codec.

The basic setup for the **Video Playback Bink** component includes the following:

- Add a **Camera** component
- Add a **Mesh** and **Video Playback Bink** component
- Configure your material

**To use the Video Playback Bink component**

1. If you don't yet have a camera in your scene, place a **Camera** (p. 566) component where your video playback is to be placed.
   
   You can use the camera to view your video playback. Ensure that the camera is facing the direction where you place your video playback component.

2. Create an entity. For more information, see Creating an Entity (p. 463).

3. Use the **Entity Inspector** (p. 475) to add a **Mesh** (p. 684) component to your entity.

4. For the **Mesh** component, select a **Mesh asset**. This is the asset that your video renders on. A cube or plane is a good test mesh.

5. Add the **Video Playback Bink** component to the same entity.
6. In the **Video Playback Bink** component, for **Video**, select the video to display.

7. For **Texture name**, enter dollar sign ($) and a name for your texture. You can enter any name, but it must begin with a $ character to indicate that it’s a render target. For example, $videotest is a valid name, but videotest isn’t.

![Video Playback Bink](image)

8. For **Frame queue ahead count**, set the number of frames to buffer.

   We recommend that you use a value from 1 to 3.

   Queueing too many frames to buffer (for example, a value of 100 frames) can use too much memory and cause performance issues.

9. Open the **Material Editor (p. 1636)**.

10. To create a material, click the **Add New Item** icon. Enter a descriptive name, such as **myvideomaterial**.

![Material Editor](image)

11. Under **Texture Maps**, on the **Diffuse** line, enter the name of your video component’s **Texture name** field. You must include the $ character.
12. Close the **Material Editor** and return to the **Entity Inspector** (p. 475). In the **Mesh** component, for the **Material override** property, select the material that you created.

You can trigger the video to play at the start of your game using Lua scripting.
Lua Bindings for Video Playback Bink

You can use Lua bindings to interact programmatically with video playback components that you've placed in your scene. Lua provides a way to establish complex logic for playing, pausing, and stopping videos.

Global Functions

The following functions provide programming interfaces for the video playback systems.

**VideoPlaybackRequestBusSender**

**Parameters**

- **EntityID**

**Return**

Returns the VideoPlaybackRequestBusSender object that is connected to the specified entity. For more information, see VideoPlaybackRequestBusSender Object (p. 917).

**VideoPlaybackNotificationBusHandler**

Exposes callbacks to your Lua script that are triggered by events during video playback.

For more information, see VideoPlaybackNotificationBusHandler Object (p. 918).

**Parameters**

- **Table** – The Lua table to which you want to expose the callback functions. Pass self to expose the callbacks to the current Lua script.

  **EntityId** –

**Return**

Returns the VideoPlaybackRequestBusSender object that is connected to the specified entity. For more information, see VideoPlaybackRequestBusSender Object (p. 917).

**VideoPlaybackRequestBusSender Object**

The VideoPlaybackRequestBusSender object contains functions with which you can send requests to the video playback component.

**Bool IsPlaying()**

Returns true if the video is playing. If the video is paused or stopped, returns false.

**Void Play()**

Plays the video. If no video is selected or the video is already playing, this has no effect.

**Void Pause()**

Pauses the video. If the video is already paused, this has no effect.

**Void Stop()**

Stops the video and remains on the last frame. When the video plays again, it begins at the first frame of the video. If the video is already stopped, this has no effect. If the video is playing or paused, the video stops.
Void EnableLooping(Bool)

Sets whether this video automatically restarts from the beginning once the end of the video is reached. Pass true to enable looping or false to disable looping. Looping is disabled by default.

Void SetPlaybackSpeed(Float)

Sets how fast the video plays. For example, 1.0 is normal speed, 0.5 is half speed, 2.0 is double speed, and so on.

Caution is advised when setting the video speed. Setting a speed that is too high can result in choppy playback.

VideoPlaybackNotificationBusHandler Object

The VideoPlaybackNotificationBusHandler object exposes callback functions to your Lua script that are triggered by events that happen during video playback.

Void OnPlaybackStarted()

Called when video playback begins.

Void OnPlaybackPaused()

Called when video playback pauses. Not called when video stops.

Void OnPlaybackStopped()

Called when video playback is stopped by the user. If the video reaches the end and is not set to loop, this function is not called.

Void OnPlaybackFinished()

Called when all frames in the video are played. This is not called if the user manually stops video playback. If looping is enabled, this function is called every time the video loops.

VisArea

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A vis area (visible area) is a shape object that helps you manage visibility and culling in a scene. When a camera is in a visible area, the camera can see only other objects that are also in the visible area. You can use the VisArea component to define indoor areas and manage overdraw. Objects are inside a visible area only if their center is inside the area.

You can also add a Portal (p. 717) component to an entity to create windows between visible areas. This helps you create beautiful areas that render efficiently.

Visible areas are only dynamic during editing. If you modify the VisArea component or move the entity that the VisArea component is attached to at runtime, the area that the VisArea component defines isn't affected. Also, the VisArea component doesn't work if it's spawned as part of a dynamic slice.
For more information, see Working with the Event Bus (EBus) system (p. 1851).

**Note**
You can't modify the OccluderArea (p. 701), Portal (p. 717), and VisArea (p. 918) components at runtime.

**Topics**
- VisArea Component Properties (p. 919)
- Vis Area Component Examples (p. 919)

### VisArea Component Properties

The VisArea component has the following properties.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>The height of the visible area.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Displays the visible area as a filled volume in the Lumberyard Editor viewport.</td>
</tr>
<tr>
<td>AffectedBySun</td>
<td>Objects in the visible area are affected by the sun.</td>
</tr>
<tr>
<td>ViewDistRatio</td>
<td>A multiplier on the view distance of the visible area during editing.</td>
</tr>
<tr>
<td>OceanIsVisible</td>
<td>Renders the ocean outside the visible area.</td>
</tr>
<tr>
<td>Edit</td>
<td>Choose Edit, and the component is locked for editing. For more information, see Editing Components in the Viewport (p. 483).</td>
</tr>
</tbody>
</table>

### Vis Area Component Examples

**Example 1**

The following example shows two boxes. One box is inside the visible area, and one is outside. From outside the visible area, the inside box isn't visible. The DisplayFilled option is enabled so that the visible area appears orange.

You can see the shadow of the box inside the visible area because the AffectedBySun option is enabled.
Example 2

In the same example, the `DisplayFilled` option is disabled in the visible area.
Example 3

In the following example, you can see the box inside the visible area, but you can't see the box outside the visible area.

You can see the ocean inside the visible area because the **OceansVisible** option is enabled.
Example: Non-Planar Legacy Vertices

If you're familiar with the legacy visible areas, you might be used to seeing the legacy visualization. In the legacy system, if you raise one vertex of the visible area higher than the other vertices, the legacy visualization displays the visible area incorrectly.

The following example is the legacy visualization for a visible area. One vertex appears higher than the others.
Example: Planar Vertices with VisArea Component

The following is an example of the VisArea component with the same vertices.
The second example is correct because the visible area is the volume on the xy-plane with a z-position and a height. This makes it faster to determine whether an object is inside the VisArea component.

The position on the z-axis is the position of the lowest point. The height is the specified height plus the difference in height between the highest and lowest point. This remains true even after transformations. If you rotate a visible area on the x- or y-axis, you might see volumes that don't make sense. All points after a transformation are projected onto the xy-plane.

When legacy visible areas are converted, Lumberyard corrects this behavior for you. All points are made planar, and any difference in height between points applies to the visible area's height.

**VR Preview**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The **VR Preview** component, when added to an entity or spawned from a slice, creates a user-editable navigation mesh (p. 686). The navigation mesh is used to define valid areas that users can teleport to. The **VR Preview** component sets up necessary dependencies for exploring or navigating a scene in virtual reality. This component contains no editable properties.

**To use the VR Preview component**

1. Create a new entity (p. 463).
2. Open the **Entity Inspector** (p. 475). Add the **VR Preview** component to the entity by clicking **Add Component, VR, VR Preview**.
   
   A navigation mesh of 50x50x50 size is generated around the entity.
3. Save the entity to a slice (p. 519).
   
   Now you can drag the saved slice to another part of the level, and the navigation mesh will be generated around the entity.

The navigation mesh that is generated is a separate entity from the original entity. Lumberyard names the new nav mesh entity by appending `_NavMesh` to the original entity name.

If you delete the original entity, the nav mesh entity continues to exist. However, if you delete the nav mesh entity, it won't be generated again.

**Water Volume**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Water Volume** component to add small and medium bodies of water to your level. Use the following properties to adjust the water volume's physical attributes, aesthetic, and complex interactions.

To use the **Water Volume** component, you must enable the Water gem for your game project and then rebuild the project. For more information, see **Enabling Gems** (p. 1064).

**Topics**

- **Water Volume Component Properties** (p. 925)
- **EBus Request Bus Interface** (p. 929)
- **Setting up Volume to Spill** (p. 943)
- **Setting up Wave Simulation** (p. 945)

**Water Volume Component Properties**

The **Water Volume** component has the following properties:

**Material**

- Specifies the water material to use. The material must have the **Water** shader added.
Minimum Spec

Sets the minimum engine specification for the volume to render.

Default: Low

Surface U Scale

Specifies how much to tile the water surface texture on the u-axis.

Default: 1.0

Range: 0.0 to # (infinity)

Surface V Scale

Specifies how much to tile the water surface texture on the v-axis.

Default: 1.0

Range: 0.0 to #

View Distance Multiplier

Adjusts the maximum view distance. The default value is 1.0. To specify a maximum view distance that is 10% further than the default, use 1.1.

Default: 1.0

Range: 0.0 to #

Fog

Density

Sets the density of the water fog.

Default: 0.5

Range: 0.0 to #

Color

Sets the color of the water fog.

Default: Deep blue (RGB 0.005, 0.01, 0.02)

Color Multiplier

Multiplies this value by the Color parameter to produce the final fog color.

Default: 1.0

Range: 0.0 to #

Color Affected by Sun

Determines whether the sky color affects the water's fog color. This property is useful for outdoor water volumes.

Default: True

Shadowing

Specifies how much shadows affect the water's fog color.

Default: 0.5
Cap at Volume Depth

Determined whether fog is rendered when the player is under the volume.

Default: False

Caustics

Enabled

Enables water caustics.

Default: True

Intensity

Affects the brightness and visibility of caustics.

Default: 1.0

Range: 0.0 to #

Tiling

Affects the number of caustics that are rendered.

Default: 1.0

Range: 0.0 to #

Height

Determines how far above the surface caustics are rendered.

Default: 0.5

Range: 0.0 to #

Advanced

Spillable Volume

Specifies the volume quantity that can spill into a container below.

Default: 0.0

Range: 0.0 to #

Volume Accuracy

Specifies the accuracy of the surface level of the spilled volume.

Default: 0.001

Range: 0.0 to #

Extrude Border

Specifies the border amount to add to the spilled volume.

Default: 0.0

Range: 0.0 to #
**Convex Border**

Determines whether the convex hull of the container should be considered.

Default: false

**Object Size Limit**

Sets the minimum volume that an object must have to affect the water surface.

Default: 0.001

Range: 0.0 to #

**Wave Simulation**

**Surface Cell Size**

Specifies the size of wave simulation cells.

Default: 0.0

Range: 0.0 to #

**Speed**

Specifies how fast each wave moves.

Default: 100.0

Range: 0.0 to #

**Dampening**

Specifies how much dampening force to apply during simulation.

Default: 0.2

Range: 0.0 to #

**Timestep**

Specifies the frequency of wave simulation ticks.

Default: 0.02

Range: 0.0 to #

**Sleep Threshold**

Sets the lowest velocity for a cell to rest.

Default: 0.01

Range: 0.0 to #

**Depth Cell Size**

Specifies the size of the depth cell.

Default: 8.0

Range: 0.0 to #

**Height Limit**

Sets the highest and lowest heights that the surface can be deformed.
Water Volume

Default: 7.0
Range: 0.0 to #

**Force**

Specifies the strength of the wave force.

Default: 10.0
Range: 0.0 to #

**Simulation Area Growth**

Adds space if the water simulation causes expansion.

Default: 0.0f
Range: 0.0 to #

**EBus Request Bus Interface**

Use the following request functions with the EBus interface to communicate with other components of your game. You can use this EBus to communicate to an entity with a **Water Volume** component attached. The EBus is available at game time and editing and can be accessed from C++, Lua, and the [Script Canvas](#) editor.

For more information about using the event bus (EBus) interface, see [Working with the Event Bus (EBus) system](#) (p. 1851).

**SetSurfaceUScale**

Sets how much to tile the surface texture horizontally (u-axis of the UV map). The surface texture is often defined as a bump or normal map.

**Parameters**

Type: Float

**Return**

None

**Scriptable**

Yes

**GetSurfaceUScale**

Returns how much to tile the surface texture horizontally (u-axis of the UV map).

**Parameters**

None

**Return**

Type: Float

**Scriptable**

Yes
**SetSurfaceVScale**

Sets how much to tile the surface texture vertically (v-axis of the UV map). The surface texture is often defined as a bump or normal map.

**Parameters**
- Type: Float

**Return**
- None

**Scriptable**
- Yes

**GetSurfaceVScale**

Returns how much to tile the surface texture vertically.

**Parameters**
- None

**Return**
- Type: Float

**Scriptable**
- Yes

**SetFogDensity**

Sets the density of the underwater fog in the volume. A lower value produces clearer water, and a higher value produces murkier water.

**Parameters**
- Type: Float

**Return**
- None

**Scriptable**
- Yes

**GetFogDensity**

Returns the fog density in the water volume.

**Parameters**
- None

**Return**
- Type: Float
Scriptable
Yes

SetFogColor
Sets the color of the underwater fog in the volume. The FogColorMultiplier parameter modifies the fog color before rendering.

Parameters
AZ::Color
Return
None
Scriptable
Yes

GetFogColor
Returns the color of the underwater fog in the volume. The FogColorMultiplier parameter does not affect the returned value.

Parameters
None
Return
AZ::Color
Scriptable
Yes

SetFogColorMultiplier
Sets the value to multiply with the FogColor parameter before rendering. A higher value produces brighter underwater fog.

Parameters
Type: Float
Return
None
Scriptable
Yes

GetFogColorMultiplier
Returns the value to multiply with the FogColor parameter before rendering.

Parameters
None
Return
Type: float
Scriptable
Yes

SetFogColorAffectedBySun
Sets whether the sun affects the underwater fog in the volume. If true, the SunColor value affects the fog color. You can set the SunColor in the Time of Day Editor. For outdoor water volumes, use true. For indoor water volumes, use false.

Parameters
Type: Boolean
Return
None
Scriptable
Yes

GetFogColorAffectedBySun
Returns whether the sun affects the underwater fog in the volume.

Parameters
None
Return
Type: Boolean
Scriptable
Yes

SetFogShadowing
Sets the strength of the shadows that fall on the water volume. If 0.0, shadows have no effect on the water volume. If 1.0, a dark shadow falls on the water volume. You must set r_FogShadowsWater to 1 for the parameter to have any effect.

Parameters
Type: Float
Return
None
Scriptable
Yes
Valid values
0.0 to 1.0
GetFogShadowing

Returns the strength of the shadows that fall on the water volume. Valid values are 0.0 to 1.0. However, this property returns the value for the FogShadowing parameter, regardless of the valid range.

Parameters

None

Return

Type: Float

Scriptable

Yes

SetCapFogAtVolumeDepth

Sets whether the underwater fog effect caps at the volume depth. If false, underwater fog renders despite the distance of the camera below the water volume.

Parameters

Type: Boolean

Return

None

Scriptable

Yes

GetCapFogAtVolumeDepth

Returns whether the underwater fog effect caps at the volume depth.

Parameters

None

Return

Type: Boolean

Scriptable

Yes

SetCausticsEnabled

Sets whether the water volume produces caustics. You must set r_WaterVolumeCaustics to 1 for the water caustic settings to have any effect. To display caustics, set the water volume to a world height of 1 or higher.

Parameters

Type: Boolean

Return

None
Scriptable
Yes

GetCausticsEnabled
Returns whether the water volume produces caustics.

Parameters
None

Return
Type: Boolean

Scriptable
Yes

SetCausticIntensity
Sets the intensity of the caustics for the water volume. This property scales the normal map when rendering to the caustic map to produce a stronger caustic effect.

Parameters
Type: Float

Return
None

Scriptable
Yes

GetCausticIntensity
Returns the intensity of the caustics for the water volume.

Parameters
None

Return
Type: Float

Scriptable
Yes

SetCausticTiling
Sets a multiplier to use for tiling the water volume's caustics. This value is multiplied with the tiling that's applied to the surface normals. You can then scale the caustic tiling separately from the material.

Parameters
Type: Float
Return

None

Scriptable

Yes

**GetCausticTiling**

Returns the value that is multiplied to the surface's normal tiling during caustic generation.

**Parameters**

None

**Return**

Type: Float

Scriptable

Yes

**SetCausticHeight**

Sets the allowable height to cast caustics above the water's surface. If 0.0, caustics render only beneath the water's surface. If you use a value greater than 0, caustics can cast on nearby surfaces that are not in the water volume.

**Parameters**

Type: Float

**Return**

None

Scriptable

Yes

**GetCausticHeight**

Returns the allowable height to cast caustics above the water's surface.

**Parameters**

None

**Return**

Type: Float

Scriptable

Yes

**SetSpillableVolume**

Sets how much volume can spill. Volumes can spill into a container but cannot spill onto terrain. A higher value allows more of the container volume to fill with water. If greater than 0, the water volume raycasts...
onto nearby geometry and attempts to spill into it. The geometry must have a concave mesh on a mesh component.

Parameters
Type: Float

Return
None

Scriptable
Yes

GetSpillableVolume
Returns how much volume can spill.

Parameters
None

Return
Type: Float

Scriptable
Yes

SetVolumeAccuracy
Sets the accuracy of the water level for the spilled water volume. The water level for spilled volume is iteratively calculated until the value is within the distance to the water level expected by the water volume. A higher value requires more iterations to calculate the water level. If 0.0, the water level iterations will reach a hard-coded limit of 100 iterations.

Parameters
Type: Float

Return
None

Scriptable
Yes

GetVolumeAccuracy
Returns the accuracy of the water level for the spilled water volume.

Parameters
None

Return
Type: Float
SetExtrudeBorder

Sets how much to increase the border of the water volume when spilled. If 0, wave simulation may cause the open edges of a water volume's surface to be visible. This is useful for wave simulation.

Parameters
   Type: Float

Return
   None

Scriptable
   Yes

GetExtrudeBorder

Returns how much to increase the border of the water volume when spilled.

Parameters
   None

Return
   Type: Float

Scriptable
   Yes

SetConvexBorder

Sets whether the water volume should look for a convex border when spilling. This is useful if the volume spills into a container that has multiple contours. Because water volumes don't support multiple contours, this property uses logic to break the volume into multiple volumes that can fit the contoured container.

Parameters
   Type: Boolean

Return
   None

Scriptable
   Yes

GetConvexBorder

Sets whether the water volume should look for a convex border when spilling.
Parameters

None

Return

Type: Boolean

Scriptable

Yes

SetObjectSizeLimit

Sets the minimum volume that is required for an object to experience volume displacement. Objects that have a volume that is smaller than this value will not deform the surface.

Parameters

Type: Float

Return

None

Scriptable

Yes

GetObjectSizeLimit

Returns the minimum volume that is required for an object to experience volume displacement.

Parameters

None

Return

Type: Float

Scriptable

Yes

SetWaveSurfaceCellSize

Sets the size of each wave simulation cell. The number of wave simulation cells is determined by the size of the volume and the individual size of the cells. This affects the water surface mesh.

- A large water volume with a small cell size produces a high number of cells and a large, complicated surface mesh. This type of mesh can make it difficult to evaluate wave simulation, unless you tune other parameters.
- A small water volume with a large cell size produces a low number of cells and a small, simple surface mesh.

Use the p_draw_helpers a_g console variable to see the tessellated surface mesh. You may need to move the editor's viewport camera closer to the volume for the mesh to render.

Parameters

Type: Float
Return
None
Scriptable
Yes

GetWaveSurfaceCellSize
Returns the size of each wave simulation cell.

Parameters
None
Return
Type: Float
Scriptable
Yes

SetWaveSpeed
Sets how fast the waves move.

Parameters
Type: Float
Return
None
Scriptable
Yes

GetWaveSpeed
Returns how fast the waves move.

Parameters
None
Return
Type: Float
Scriptable
Yes

SetWaveDampening
Sets how much to dampen the wave simulation. A higher value produces waves that lose velocity more quickly.
GetWaveDampening

Returns how much to dampen the wave simulation.

Parameters

None

Return

Type: Float

Scriptable

Yes

SetWaveTimestep

Sets how often the wave simulation ticks. A lower value produces frequent simulation ticks, which resolves a simulation quickly.

Parameters

Type: Float

Return

None

Scriptable

Yes

GetWaveTimestep

Returns how often the wave simulation ticks.

Parameters

None

Return

Type: Float

Scriptable

Yes
SetWaveSleepThreshold

Sets the minimum velocity that is required for a cell to sleep. A cell is at rest and no longer applies force to its neighbors when the velocity reaches this threshold.

Parameters

  Type: Float

Return

  None

Scriptable

  Yes

GetWaveSleepThreshold

Returns the minimum velocity that is required for a cell to sleep.

Parameters

  None

Return

  Type: Float

Scriptable

  Yes

SetWaveDepthCellSize

Sets the size of each depth cell in a wave simulation. The number of wave simulation cells is determined by the size of the volume and the individual size of the cells. A smaller cell size produces more depth cells and an intensive wave simulation.

Parameters

  Type: Float

Return

  None

Scriptable

  Yes

GetWaveDepthCellSize

Returns the size of each depth cell in a wave simulation.

Parameters

  None

Return

  Type: Float
SetWaveHeightLimit

Sets the height and depth limit for generated waves. A height limit of 3 means the cell deforms up three units and down three units.

Parameters

Type: Float

Return

None

Scriptable

Yes

GetWaveHeightLimit

Returns the height and depth limit for generated waves.

Parameters

None

Return

Type: Float

Scriptable

Yes

SetWaveForce

Sets the amount of force that wave simulation cells apply to each other. A low wave force produces high resistance between cells. A high wave force produces less resistance and transmits more energy between cells. More transmitted energy results in larger, more pronounced waves.

Parameters

Type: Float

Return

None

Scriptable

Yes

GetWaveForce

Returns the amount of force that wave simulation cells apply to one another.

Parameters

None
Return
Type: Float
Scriptable
Yes

SetWaveSimulationAreaGrowth
Sets the size boundary to add to a simulation area. This determines where the water volume simulation can move and grow outside the initial defined area. Increase the variable to add the size boundary to the simulation area. The p_draw_helpers_a_gj console variable draws the wave simulation area as a checkerboard height field.

Parameters
None
Return
None
Scriptable
Yes

GetWaveSimulationAreaGrowth
Returns the size boundary to add to a simulation area.

Parameters
None
Return
None
Scriptable
Yes

Example Water Volume Component Request Bus

```lua
function example:OnActivate
    local waveForce = WaterVolumeComponentRequestBus.Event.GetWaveForce(self.entityId)
    WaterVolumeComponentRequestBus.Event.SetWaveForce(self.entityId, waveForce * 10)
end
```

Setting up Volume to Spill

You can set water volumes to spill into container volumes. If the Spillable Volume parameter is a value above 0, the volume raycasts down and searches for a suitable container to spill into. This includes any mesh component with a concavity. A higher value for the Spillable Volume parameter produces a higher water level in the target container.

The spilled water volume behaves similarly to the source water volume and can simulate waves. The spilled volume is calculated iteratively, and the Volume Accuracy parameter determines the accuracy of
the water level. Spilled water volumes are calculated once and should not incur a per-frame performance penalty due to volume calculation.

The following image shows water volume above a pool that does not allow spilling.

The following image shows the water volume spilling 100 units into the pool.

The following image shows the water volume spilling 1000 units into the pool.
Setting up Wave Simulation

You can use wave simulation to create the appearance of the water surface reacting to physical forces. This feature does not simulate ocean waves. Because wave simulation is an expensive task, consider the performance impact before you enable this feature.

To enable wave simulation on a water volume, set the **Surface Cell Size** parameter to a value above 0. For smaller water volumes, you can use a value of 2 or 3. A smaller cell size produces more cells and a longer evaluation time for the simulation.

Even without wave simulation the **Water** shader deforms the surface mesh to make the water appear in motion.

**Example**

The following is an example **Water Volume** component.
White Box component

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The White Box component is a tool you can use to sketch 3D proxy meshes in Lumberyard Editor. Add the White Box component to an entity, select a primitive shape to use as a basis for your proxy mesh, then enter edit mode to access the tools to quickly rough out a mesh for your entity.

Because White Box is implemented as a component, you can create well-defined entities that accurately represent the size, shape, and function of the final production entity in Lumberyard Editor before investing the time and effort into building finished models for your entity. Meshes created with White Box can be saved to disk as white box mesh assets (.wbm) and reused in other White Box components. White box meshes can also be exported to .obj files and used as a template in a third-party 3D modeling application to build final production assets.

Topics
- White Box properties (p. 947)
- White Box edit mode (p. 948)
White Box properties

![White Box properties](image)

**Default Shape**

The default primitive shape of the white box mesh. From the list you can select a primitive shape, or choose to load a saved white box mesh (.wbm) asset. The default primitive size is one meter in world space.

**White Box default shapes:**

- Cube
- Tetrahedron
- Icosahedron
- Cylinder
- Sphere
- Custom Mesh Asset

**Note**

When Custom Mesh Asset is selected, a Mesh Asset file field appears in the component interface below Default Shape that you can use to select a saved white box mesh (.wbm) asset.

**Save as asset**

Save the proxy mesh to a white box mesh (.wbm) asset. You can load the saved (.wbm) asset in other White Box components. The .wbm file functions like an instance, and any changes made to the mesh propagate to all White Box components that use the .wbm file.
Tint

Set a tint color for the white box mesh. Choose the swatch to open a color picker, or enter comma separated red, blue, and green 8-bit values into the field to set a tint color for the **White Box** component.

Use Texture

Enable **Use Texture** to display a checkerboard texture on the white box mesh. Each square is a half meter in size, and the texture is projected on the local X, Y, and Z axes of the mesh. This maintains an easy reference for the size of the proxy mesh, regardless of how the entity is oriented in the level.

Visible

Enable **Visible** to make the white box mesh visible at runtime. When you use White Box to create custom invisible collision meshes, disable the **Visible** property to hide the mesh at runtime.

Edit

Choose this button to enter edit mode and modify the white box mesh. For information on editing the white box mesh, see **White Box edit mode** (p. 948). Choose this same button (labeled **Done**) to exit edit mode.

Export

Export the mesh to a .obj file. The .obj file can be loaded into a 3D modeling application and used as a template for creating the production mesh asset for the entity.

**White Box edit mode**

In edit mode, you can quickly sketch meshes for your entities in Lumberyard Editor by selecting and dragging the face, edge, and vertex components of the white box mesh. To begin, add a **White Box**
component to an entity, choose a default primitive shape in the White Box component interface, and choose Edit to enter edit mode.

**Move**

**Move polygon**

1. Hover over a polygon.
2. Hold the left mouse button.
3. Drag the polygon along its normal.

**Move edge**

1. Hover over an edge.
2. Hold the left mouse button.
3. Drag the edge.
Move vertex

1. Hover over a vertex. The vertex will highlight.
2. Hold the left mouse button.
3. Drag the vertex along the edge guides.
Scale

Scale polygon

1. Select (left click) a polygon.
2. Hover over one of the polygon's vertices.
3. Hold the left mouse button.
4. Drag the vertex toward or away from the center of the selected polygon.

Scale edge

1. Select (left click) an edge.
2. Hover over one of the edge's vertices.
3. Hold the left mouse button.
4. Drag the vertex along the length of the selected edge.
Non-uniform scale edge

1. Select (left click) an edge.
2. Hover over one of the edge's vertices.
3. Hold Alt and drag the vertex along the length of the selected edge.
Extrude

**Extrude polygon**

1. Hover over a polygon.
2. Hold **Ctrl** and left mouse button.
3. Drag the polygon along its normal.
Extrude edge

1. Hover over an edge.
2. Hold Ctrl and left mouse button.
3. Drag the edge.
Extrude scale

1. Select (left click) a polygon.
2. Hover over one of the polygon's vertices.
3. Hold Ctrl and left mouse button.
4. Drag the vertex toward or away from the center of the selected polygon to scale.
5. Hover over the selected polygon.
6. Hold the left mouse button.
7. Drag the polygon along its normal.
**Flip edge**

1. Hold **Ctrl + Shift** to show hidden edges.
2. Right click on a hidden edge to flip its orientation.
Hide/Show edge

Hide an edge

1. Select (left click) an edge.
2. Press H.
Show edge

1. Hold Ctrl and Shift to show edges.
2. Select (left click) an edge to unhide it.
Hide/Show vertex

Hide a vertex

1. Select (left click) a vertex.
2. Press H.
Show vertex

1. Hold Ctrl and Shift to show hidden vertices.
2. Select (left click) a vertex to unhide it.

White Box Collider component

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can enable collision on white box meshes in Lumberyard by adding the White Box Collider component to an entity that has a White Box component mesh. The White Box Collider component supports collision layers and physics materials. It can be used with static and kinematic white box meshes. The White Box Collider component uses the white box mesh as the collision surface. Unlike the PhysX Collider component, there is no need to specify a collision shape or provide a PhysX mesh asset.
In the animation above, the **White Box Collider** component is applied to a static **White Box** component. The **White Box** component can be edited and the changes tested for collision immediately.
In the animation above, the door was created with White Box and animated with a script. Note when
the White Box component is edited, the White Box Collider component automatically recognizes the
changes to the mesh. You can then test the changes immediately.

**White Box Collider properties**

<table>
<thead>
<tr>
<th>Collision Layer</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collides With</td>
<td>All</td>
</tr>
<tr>
<td><strong>Physics Material</strong></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>surfacetype materia library (default)</td>
</tr>
<tr>
<td><strong>Mesh Surfaces</strong></td>
<td>1 elements</td>
</tr>
<tr>
<td>Entire object</td>
<td>Default</td>
</tr>
<tr>
<td><strong>Tag</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Body Type</strong></td>
<td>Static</td>
</tr>
</tbody>
</table>
Collision Layer
The collision layer that's assigned to the collider. For more information, see Collision Layers (p. 2785).

Collides With
The collision group containing the layers that this collider collides with. For more information, see Collision Groups (p. 2787).

Physics Material - Library
Set the physics material library for this collider.

Physics Material - Mesh Surfaces
Choose a material from the physics material library for this collider. The material is applied to the entire white box entity.

Tag
Set a tag for this collider. Tags can be used to quickly identify components in script or code.

Body Type
Select Static for non-moving entities. Select Kinematic for animated entities.

Note
The White Box collider must be set to Static to interact with the PhysX Character Controller.

Converting Entities with the Legacy Converter

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Legacy Converter converts your legacy entities (CryEntities) to the new component entity system. Legacy features will eventually be removed from Lumberyard Editor. When you use the Legacy Converter, it tries to convert all legacy entities in your level. You cannot select which entities to convert or skip. Once an entity is converted, it cannot be converted back to a legacy entity.

Topics
- Converting Your Legacy Entities (p. 963)
- List of Legacy Entity Conversions (p. 964)

Converting Your Legacy Entities
If your level has legacy entities, Lumberyard Editor detects them and prompts you to convert them.

To convert your legacy entities
1. Do any of the following to open the Legacy Converter:
   - If the CryEntity Removal gem is enabled, and you load a level that contains legacy entities, the Legacy Converter opens by default.
List of Legacy Entity Conversions

The Legacy Converter converts each legacy entity to its corresponding component entity. For a list of legacy entities, see the Entity Reference in the Amazon Lumberyard Legacy Reference.

Converted entities keep the same position in the viewport. For more information, see Using the Viewport (p. 207).
Note
Some legacy entities cannot be converted at this time.

See the following tables for the converted component entity.

Contents
- Area Box Entity (p. 965)
- Area Sphere Entity (p. 965)
- Brush, Geom, and Simple Entities (p. 965)
- Camera Entity (p. 966)
- Camera Target Entity (p. 966)
- Comment Entity (p. 966)
- Environment Probe Entity (p. 967)
- Decal Entity (p. 968)
- Designer Objects (p. 969)
- Groups and Layers (p. 970)
- Light Entity (p. 971)
- Light Entity with Lens Flare (p. 972)
- Particle Effect Entity (p. 972)
- Proximity Trigger Entity (p. 973)
- Tag Point Entity (p. 973)

Area Box Entity
An area box entity is converted to a component entity that is attached with a Box Shape (p. 818) component.

<table>
<thead>
<tr>
<th>Settings in Area Box Entity</th>
<th>Converted Settings in Box Shape Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Dimensions : x</td>
</tr>
<tr>
<td>Length</td>
<td>Dimensions : y</td>
</tr>
<tr>
<td>Height</td>
<td>Dimensions : z</td>
</tr>
</tbody>
</table>

Area Sphere Entity
An area sphere entity is converted to a component entity that is attached with a Sphere Shape (p. 818) component.

<table>
<thead>
<tr>
<th>Settings in Area Sphere Entity</th>
<th>Converted Settings in Sphere Shape Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td>Radius</td>
</tr>
</tbody>
</table>

Brush, Geom, and Simple Entities
The brush, geom, and simple entities are converted to a component entity that is attached with a Mesh (p. 684) component or Skinned Mesh component, depending on the asset type.
List of Legacy Entity Conversions

<table>
<thead>
<tr>
<th>Settings in Brush, Geom, and Simple Entities</th>
<th>Converted Settings in Mesh or Skinned Mesh Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry (set to .cdf files)</td>
<td>Character Definition (Skinned Mesh component)</td>
</tr>
<tr>
<td>Geometry (set to non .cdf file)</td>
<td>Static asset (Mesh component)</td>
</tr>
</tbody>
</table>

**Camera Entity**

A camera entity is converted to a component entity that is attached with a [Camera (p. 566)](#) component.

<table>
<thead>
<tr>
<th>Settings in Camera Entity</th>
<th>Converted Settings in Camera Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOV</td>
<td>Field of View</td>
</tr>
<tr>
<td>NearZ</td>
<td>Near Clip Distance</td>
</tr>
<tr>
<td>FarZ</td>
<td>Far Clip Distance</td>
</tr>
</tbody>
</table>

**Camera Target Entity**

The camera target entity is converted to an empty component entity.

**Comment Entity**

A Comment entity is converted to a component entity that is attached with a [Camera (p. 566)](#) component.

- Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Comment component allows you to add long-form text comments for component entities. When enabled, the Comment component displays a dialog box that expands based on the size of the comment that you enter. The following examples demonstrate how you can use the comment text box:

- Explain how the scripts or components on an entity interact with other scripts or components
- Describe how everything in a level ties together
- Send descriptions, instructions, or notes to team members
Comment Properties

The Comment component has the following property:

- Comment text box: Stores the user comment for the component entity.

Default: None

Using the Comment Component

You can use this feature by adding the component to an entity in your level.

To use the Comment component:

1. In Lumberyard Editor, right-click the viewport in your level, and click Create entity.
2. In the Entity Inspector, click Add Component.
3. Under Editor, click Comment.
4. In the Entity Inspector, under Comment, add comments for the component entity in the text box.

<table>
<thead>
<tr>
<th>Settings in Comment Entity</th>
<th>Converted Settings in Comment Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Comment</td>
</tr>
</tbody>
</table>

Environment Probe Entity

The environment probe entity is converted to a component entity that is attached with an Environment Probe (p. 587) component.

<table>
<thead>
<tr>
<th>Settings in Environment Probe Entity</th>
<th>Converted Settings in Environment Probe Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnvironmentProbe Params : cubemap_resolution</td>
<td>Environment Probe Settings : Resolution</td>
</tr>
<tr>
<td>EnvironmentProbe Params : Outdoor Only</td>
<td>Options : Indoor only</td>
</tr>
</tbody>
</table>
### List of Legacy Entity Conversions

#### Settings in Environment Probe Entity

| EnvironmentProbe Params : ViewDistanceMultiplier | Options : View distance multiplier |
| EnvironmentProbe Properties : Active | On initially |
| EnvironmentProbe Properties : BoxSizeX | Environment Probe Settings : Area dimensions : X |
| EnvironmentProbe Properties : BoxSizeY | Environment Probe Settings : Area dimensions : Y |
| EnvironmentProbe Properties : BoxSizeZ | Environment Probe Settings : Area dimensions : Z |
| Color : Diffuse | General Settings : Color |
| Color : DiffuseMultiplier | General Settings : Diffuse multiplier |
| Color : SpecularMultiplier | General Settings : Specular multiplier |
| Options : AffectVolumetricFogOnly | Options : Volumetric fog only |
| Options : AttenuationInFallOfMax | Environment Probe Settings : Attenuation fall off |
| Options : IgnoreVisAreas | Options : Ignore vis areas |
| Options : Sort Priority | Environment Probe Settings : Sort priority |
| Options : VolumetricFog | Options : Volumetric fog |
| OptionsAdvanced : deferred_cubemap | Cubemap generation: Cubemap asset |
| Projection : BoxHeight | Environment Probe Settings : Box height |
| Projection : BoxLength | Environment Probe Settings : Box length |
| Projection : BoxProject | Environment Probe Settings : Box projected |
| Projection : BoxWidth | Environment Probe Settings : Box width |

#### Decal Entity

A decal entity is converted to a component entity that is attached with a **Decal (p. 585)** component.

<table>
<thead>
<tr>
<th>Settings in Decal Entity</th>
<th>Converted Settings in Decal Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity : Mtl</td>
<td>Decal Settings : Material</td>
</tr>
<tr>
<td>Entity : MinSpec</td>
<td>Decal Settings : Minimum spec</td>
</tr>
<tr>
<td>Entity Params : ProjectType</td>
<td>Decal Settings : Projection type</td>
</tr>
<tr>
<td>Entity Params : Deferred</td>
<td>Decal Settings : Deferred</td>
</tr>
<tr>
<td>Entity Params : View Distance Multiplier</td>
<td>Options : View distance multiplier</td>
</tr>
<tr>
<td>Entity Params : SortPriority</td>
<td>Decal Settings : View distance multiplier</td>
</tr>
<tr>
<td>Entity Params : Projection Depth (Deferred)</td>
<td>Decal Settings : Depth</td>
</tr>
</tbody>
</table>
Designer Objects

The Legacy Converter converts designer objects to a component entity that is attached with the following components:

- **Mesh** (p. 684)
- **Static Physics** component.
- **Mesh Collider**

The Legacy Converter converts the original shape of designer objects into .cgf files and then saves them to your game project directory.

<table>
<thead>
<tr>
<th>Settings in Designer Object</th>
<th>Converted Settings in Mesh Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>Mesh asset (.cgf file)</td>
</tr>
</tbody>
</table>

**Example**

1. You have a legacy designer object named *Designer1* shaped as a sphere.

2. With the Legacy Converter, you convert the object into a entity with the **Mesh**, **Static Physics**, and **Mesh Collider** components attached.

3. The Legacy Converter creates a .cgf file and saves it to the `lumberyard_version\dev\game-project\Objects\DesignerConversion\level-name` directory.
4. In the **Mesh** component, the **Mesh asset** property specifies this `Designer1.cgf` file. The entity has the same shape as the original designer object.

**Note**
You cannot edit a designer object after it is converted into a component entity. You can use the legacy **Designer Tool** to create your object and convert it again. For more information, see Using the Designer Tool in the Amazon Lumberyard Legacy Reference.

### Groups and Layers

Groups and layers are converted to empty component entities with the same names.

Legacy entities that are nested under legacy groups or layers appear nested under the newly created component entities, which keep the same hierarchy.

If a legacy entity belongs to a layer and a group, the converted component entity appears under the converted group component entity. It also appears as a child of the converted layer component entity.

**Example**

1. You have the following legacy entities in your level:
   - LegacyEntityA belongs to LegacyLayerA, so the hierarchy appears as: LegacyLayerA [LegacyEntityA]
2. You use the Legacy Converter to convert your entities.
3. The converted entities have the following hierarchy:

   ```
   ```
**Light Entity**

A light entity with the **Planar Light** setting specified is converted to a component entity that is attached with a **Area Light (p. 546)** component.

A light entity with the **Projector** setting specified to **Texture** is converted to a component entity attached with a **Projector Light (p. 723)** component.

A light entity with neither the **Planar Light** or **Projector** setting specified to **Texture** is converted to a component entity attached with a **Point Light (p. 706)** component

<table>
<thead>
<tr>
<th>Settings in Light Entity</th>
<th>Converted Settings Common in Area Light, Projector Light, and Point Light Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Params : Outdoor Only</td>
<td>Options : Indoor Only</td>
</tr>
<tr>
<td>Entity Params : ViewDistanceMultiplier</td>
<td>Options : View distance multiplier</td>
</tr>
<tr>
<td>Entity Params : HiddenInGame</td>
<td>Visible</td>
</tr>
<tr>
<td>Entity Properties : Active</td>
<td>On initially</td>
</tr>
<tr>
<td>Entity Properties : Color : Diffuse</td>
<td>General Settings : Color</td>
</tr>
<tr>
<td>Entity Properties : Color : DiffuseMultiplier</td>
<td>General Settings : Diffuse multiplier</td>
</tr>
<tr>
<td>Entity Properties : Color : SpecularMultiplier</td>
<td>General Settings : Specular multiplier</td>
</tr>
<tr>
<td>Entity Properties : Options : AffectsThisAreaOnly</td>
<td>Options : Affects this area only</td>
</tr>
<tr>
<td>Entity Properties : Options : AffectsVolumetricFogOnly</td>
<td>Options : Volumetric fog only</td>
</tr>
<tr>
<td>Entity Properties : Options : Ambient</td>
<td>General Settings : Ambient</td>
</tr>
<tr>
<td>Entity Properties : Options : IgnoresVisArea</td>
<td>Options : Ignore vis areas</td>
</tr>
<tr>
<td>Entity Properties : Options : VolumetricFog</td>
<td>Options : Volumetric fog</td>
</tr>
<tr>
<td>Entity Properties : Style : LightStyle</td>
<td>Animation : Style</td>
</tr>
<tr>
<td>Entity Properties : Style : AnimationSpeed</td>
<td>Animation : Speed</td>
</tr>
<tr>
<td>Entity Properties : Style : AnimationPhase</td>
<td>Animation : Phase</td>
</tr>
<tr>
<td>Entity Properties : Shadows : CastShadows (spec)</td>
<td>Options : Cast shadow spec</td>
</tr>
<tr>
<td>The following settings appear only in the <strong>Entity Inspector</strong> when the setting <strong>Cast shadow spec</strong> has any value, expect <strong>Never</strong>.</td>
<td></td>
</tr>
<tr>
<td>Entity Properties : Shadows : ShadowBias</td>
<td>Shadow Settings : Shadow bias</td>
</tr>
<tr>
<td>Entity Properties : Shadows : ShadowResolutionScale</td>
<td>Shadow Settings : Shadow resolution scale</td>
</tr>
<tr>
<td>Entity Properties : Shadows : ShadowSlopeBias</td>
<td>Shadow Settings : Shadow slope bias</td>
</tr>
<tr>
<td>Entity Properties : Shadows : ShadowUpdateMinRadius</td>
<td>Shadow Settings : Shadow update radius</td>
</tr>
</tbody>
</table>
## List of Legacy Entity Conversions

### Settings in Light Entity | Converted Settings in Area Light, Projector Light, and Point Light Components
---
Entity Properties: Shadows: ShadowUpdateRatio | Shadow Settings: Shadow update ratio

### Settings in Light Entity | Converted Settings in Projector Light Component
---
AttenuationBulbSize | Projector Light Settings: Attenuation bulb size
Projector: ProjectNearPlane | Projector Light Settings: Near plane
Projector: ProjectorFov | Projector Light Settings: FOV
Projector: Texture | Projector Light Settings: Texture
Radius | Projector Light Settings: Max distance

### Settings in Light Entity | Converted Settings in Area Light Component
---
Radius | Area Light Settings: Max distance
Shape: SourceDiameter | Area Light Settings: Area height
Shape: SourceWidth | Area Light Settings: Area width

### Settings in Light Entity | Converted Settings in Point Light Component
---
AttenuationBulbSize | Point Light Settings: Attenuation bulb size
Radius | Point Light Settings: Max distance

### Light Entity with Lens Flare
A light entity with a lens flare is converted to a component entity that is attached with a Lens Flare component and a light component.

The lens flare asset that is specified for a light entity is converted to the Lens Flare component settings: Library and Lens flare.

### Particle Effect Entity
The particle effect entity is converted to a component entity that is attached with a Particle component.
**Settings in Particle Effect Entity**

<table>
<thead>
<tr>
<th>ParticleEntity Properties</th>
<th>Spawn Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime</td>
<td>Pre-Roll</td>
</tr>
<tr>
<td>CountScale</td>
<td>Count scale</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Time scale</td>
</tr>
<tr>
<td>PulsePeriod</td>
<td>Pulse period</td>
</tr>
<tr>
<td>Scale</td>
<td>Speed scale</td>
</tr>
<tr>
<td>Strength</td>
<td>Strength curve time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Audio</th>
<th>Audio Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableAudio</td>
<td>Enable audio</td>
</tr>
<tr>
<td>Rtpc</td>
<td>Audio RTPC</td>
</tr>
</tbody>
</table>

**Proximity Trigger Entity**

A proximity trigger entity is converted to a component entity attached with a **Box Shape (p. 818)** component, and a **Trigger Area (p. 890)** component.

<table>
<thead>
<tr>
<th>Settings in Proximity Trigger Entity</th>
<th>Converted Settings in Box Shape Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Properties : DimX</td>
<td>Dimensions : x</td>
</tr>
<tr>
<td>Entity Properties : DimY</td>
<td>Dimensions : y</td>
</tr>
<tr>
<td>Entity Properties : DimZ</td>
<td>Dimensions : z</td>
</tr>
</tbody>
</table>

**Tag Point Entity**

A tag point entity is converted to an empty component entity.

---

**Programmer's Guide to Entities and Components**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This guide provides engine and game programmers with examples and best practices for creating and reflecting custom Lumberyard components in C++.

For information on using the component entity system in Lumberyard Editor, see Working with component entities (p. 462).

For C++ API reference documentation on the component entity system, see the Amazon Lumberyard C++ API Reference.

Lumberyard uses a lightweight entity/component model called the component entity system for both game objects and systems. Lumberyard entities are simply an ID and a container of components. They have no functionality associated with them. Lumberyard's component model is granular: It expects each...
Creating a Component

A component in Lumberyard is a simple class that inherits from Lumberyard's `AZ::Component`. A component's behavior is determined by its reflected data and the actions it takes when it is activated. This section shows you how to create Lumberyard components programmatically. For information about

Topics

- Creating a Component (p. 974)
- Registering Your Component (p. 977)
- Reflecting a Component for Serialization and Editing (p. 977)
- Defining and Using Component Services (p. 983)
- Editor Components (p. 984)
- Creating System Components (p. 987)
- Programmer's Guide to Component Mode (p. 992)
- Components and EBuses (p. 1015)
- Tick Bus and Components (p. 1019)
- Exposing Custom Components to Track View for Animation (p. 1021)
- Components and EBuses: Best Practices (p. 1026)
adding and customizing the components available in Lumberyard Editor, see Working with component entities (p. 462).

**Component Example**

An example component class skeleton follows:

```cpp
#include <AzCore/Component/Component.h>

class MyComponent
    : public AZ::Component
{
 public:
    AZ_COMPONENT(MyComponent, "{0C09F774-DECA-40C4-8B54-3A93033EC381}", AZ::Component);

    // AZ::Component interface implementation.
    void Init() override            {}
    void Activate() override      {}
    void Deactivate() override      {}

    // Required Reflect function.
    static void Reflect(AZ::ReflectContext* context);

    // Optional functions for defining provided and dependent services.
    static void GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided);
    static void GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent);
    static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required);
    static void GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible);
};
```

**Component Members**

The required and optional members that a component comprises are as follows:

AZ::Component

Every component must include AZ::Component somewhere in its inheritance ancestry. Noneditor components generally inherit directly from AZ::Component, as in the following example:

```cpp
class MyComponent
    : public AZ::Component
{
}
```

You can also create your own component class hierarchies.

AZ_COMPONENT Macro

Every component must specify the AZ_COMPONENT macro in its class definition. The macro takes two arguments:

1. The component type name.
2. A unique UUID. You may use any UUID generator to produce the value. Visual Studio provides this functionality through Tools, Create GUID. Use the Registry Format setting, and then copy and paste the value that is generated.

A sample AZ_COMPONENT macro follows:

```cpp
AZ_COMPONENT(MyComponent, "{0C09F774-DECA-40C4-8B54-3A93033EC381}", AZ::Component);
```
AZ::Component Functions

To define a component's behavior, you generally override three AZ::Component functions: Init, Activate, and Deactivate:

```cpp
void Init() override {}  // Optional
void Activate() override {}  // Required
void Deactivate() override {}  // Required
```

These functions are as described as follows:

**Init()**

(Optional) Called only once for a given entity. It requires minimal construction or setup work, since the component may not be activated anytime soon. An important best practice is to minimize your component's CPU and memory overhead while the component is inactive.

**Activate()**

(Required) Called when the owning entity is being activated. The system calls your component's Activate() function only if all dependent or required services are present. Your Activate function is always called after any components that it depends on. In addition, the component makeup of an entity never changes while the entity is active. Consequently, it is safe to cache pointers or references to other components on the entity when performance is critical.

**Deactivate()**

(Required) Called when the owning entity is being deactivated. The order of deactivation is the reverse of activation, so your component is deactivated before the components it depends on. As a best practice, make sure your component returns to a minimal footprint when it is deactivated. In general, deactivation should be symmetric to activation.

*Note*

Destruction does not necessarily follow deactivation. An entity can be deactivated and then activated again without being destroyed, so ensure that your components support this efficiently. However, when you do destroy your entity, Lumberyard ensures that your Deactivate() function is called first. Components must be authored with this in mind.

**Reflect()**

(Required) All components are AZ reflected classes. Because all components must be serializable and editable, they must contain a Reflect() function, as in the following example:

```cpp
// Required Reflect function.
static void Reflect(AZ::ReflectContext* context);
```

For more information, see Reflecting a Component for Serialization and Editing (p. 977).

Logical Services

(Optional) Components can define any combination of logical services that they provide, depend on, require, or are incompatible with. To define these logical services, use the following functions:

```cpp
// Optional functions for defining provided and dependent services.
static void GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided);
static void GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent);
static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required);
static void GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible);
```
Registering Your Component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You register a component during application startup. You implement the registration in the AZ::Module class of the module in which the component is written.

The following example module creates descriptors for the components that are declared within the module.

```cpp
//MyModule.cpp
MyModule::MyModule()
{
    // Create descriptors for components declared within this module.
    m_descriptors.insert(m_descriptors.end(), {
        MyComponent::CreateDescriptor(),
    });
}
```

For more information, see the The Module Class (p. 1093) section of the Using AZ Modules to Initialize Gems (p. 1090).

Reflecting a Component for Serialization and Editing

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Components use AZ reflection to describe the data they serialize and how content creators interact with them.

The following example reflects a component for serialization and editing:

```cpp
class MyComponent
    : public AZ::Component
{
    // ... AZ_COMPONENT, Activate(), Deactivate(), etc, ...
    static void Reflect(AZ::ReflectContext* context);

    enum class SomeEnum
    {
        EnumValue1,
        EnumValue2,
    }
    float m_someFloatField;
    AZStd::string m_someStringField;
    SomeEnum m_someEnumField;
    AZStd::vector<SomeClassThatSomeoneHasReflected> m_things;
    int m_runtimeStateNoSerialize;
}

void MyComponent::Reflect(AZ::ReflectContext* context)
```

Version 1.28
977
AZ::SerializeContext* serialize = azrtti_cast<AZ::SerializeContext*>(context);
if (serialize)
{
    // Reflect the class fields that you want to serialize.
    // In this example, m_runtimeStateNoSerialize is not reflected for serialization.
    // Base classes with serialized data should be listed as additional template
    // arguments to the Class< T, ... >() function.
    serialize->Class<MyComponent, AZ::Component>()
        ->Version(1)
        ->Field("SomeFloat", &MyComponent::m_someFloatField)
        ->Field("SomeString", &MyComponent::m_someStringField)
        ->Field("Things", &MyComponent::m_things)
        ->Field("SomeEnum", &MyComponent::m_someEnumField);  
    AZ::EditContext* edit = serialize->GetEditContext();
    if (edit)
    {
        edit->Class<MyComponent>("My Component", "The World's Most Clever Component")
            ->ClassElement(AZ::Edit::ClassElements::EditorData, "")
            ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu, AZ_CRC("Game"))
            ->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someFloatField, "Some Float", "This is a float that means X.")
            ->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someStringField, "Some String", "This is a string that means Y.")
            ->DataElement(AZ::Edit::UIHandlers::ComboBox, &MyComponent::m_someEnumField, "Choose an Enum", "Pick an option among a set of enum values.")
                ->EnumAttribute(MyComponent::SomeEnum::EnumValue1, "Value 1")
                ->EnumAttribute(MyComponent::SomeEnum::EnumValue2, "Value 2")
            ->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_things, "Bunch of Things", "A list of things for doing Z.");
    }
}

The preceding example adds five data members to MyComponent. The first four data members will be serialized. The last data member will not be serialized because it contains only the runtime state. This is typical; components commonly contain some members that are serialized and others that are not.

It is common for fields to be reflected for serialization, but not for editing, when using advanced reflection features such as change callbacks (p. 981). In these cases, components may conduct complex internal calculations based on user property changes. The result of these calculations must be serialized but not exposed for editing. In such a case, you reflect the field to SerializeContext but do not add an entry in EditContext. An example follows:

serialize->Class<MyComponent>()
    ->Version(1)
    ...
    ->Field("SomeFloat", &MyComponent::m_someFloatField)
    ->Field("MoreData", &MyComponent::m_moreData)
    ...
...
AZ::EditContext* edit = serialize->GetEditContext();
if (edit)
{
    edit->Class<MyComponent>("My Component", "The World's Most Clever Component")
        ->ClassElement(AZ::Edit::ClassElements::EditorData, "")
}
Reflecting a Component for Serialization and Editing

```cpp
->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu, AZ_CRC("Game"))
->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someFloatField, "Some Float", "This is a float that means X.")
->Attribute(AZ::Edit::Attributes::ChangeNotify, &MyComponent::CalculateMoreData)
// m_moreData is not reflected for editing directly.
```

Lumberyard has reflection contexts for different purposes, including the following:

- **Serialization Context** (p. 1033) – Contains reflection data for serialization and construction of objects.
- **Edit Context** (p. 1052) – Contains reflection data for visual editing of objects, as in Lumberyard Editor.
- **Behavior Context** (p. 1053) – Contains reflection for runtime manipulation of objects from Lua, Script Canvas (p. 2420), or other external sources.
- **NetworkContext** – Contains reflection for networking purposes, including marshaling, quantization, and extrapolation.

**Note**
This topic covers only SerializeContext and EditContext.

All of Lumberyard's reflection API operations are designed to be simple, human readable, and human writable, with no forced dependency on code generation.

A component's `Reflect()` function is invoked automatically for all relevant contexts.

The following code dynamically casts the anonymous context provided to a serialize context, which is how components discern the type of context that `Reflect()` is being called for.

```cpp
AZ::SerializeContext* serialize = azrtti_cast<AZ::SerializeContext*>(context);
```

**Serialization**

Reflecting a class for serialization involves a builder pattern style markup in C++, as follows:

```cpp
serialize->Class<TestAsset>()
->Version(1)
->Field("SomeFloat", &MyComponent::m_someFloatField)
->Field("SomeString", &MyComponent::m_someStringField)
->Field("Things", &MyComponent::m_things)
->Field("SomeEnum", &MyComponent::m_someEnumField)
```

The example specifies that `m_someFloatField`, `m_someStringField`, `m_things`, and `m_someEnumField` should all be serialized with the component. Field names must be unique and are not user facing.

**Tip**
We recommend that you keep field names simple for future proofing. If your component undergoes significant changes and you want to write a data converter to maintain backward data compatibility, you must reference the field names directly.

The preceding example reflects two primitive types—a float, and a string—as well as a container (vector) of some structure. AZ reflection, serialization, and editing natively support a wide variety of types:

- Primitive types, including integers (signed and unsigned, all sizes), floats, and strings
- Enums
Lumberyard User Guide Lumberyard User Guide
Reﬂecting a Component for Serialization and Editing

• AZStd containers (ﬂat and associative), including AZStd::vector, AZStd::list, AZStd::map,
AZStd::unordered_map, AZStd::set, AZStd::unordered_set, AZStd:pair, AZStd::bitset,
AZStd::array, ﬁxed C-style arrays, and others.
• Pointers, including AZStd::smart_ptr, AZStd::intrusive_ptr, and raw native pointers.
• Any class or structure that has also been reﬂected.

Note

The example omits the reﬂection code for SomeClassThatSomeoneHasReflected. However,
you need only reﬂect the class. After that, you can freely reﬂect members or containers of that
class in other classes.
For C++ API reference documentation on the serialize context, see the SerializeContext Class Reference in
the Amazon Lumberyard C++ API Reference.

Editing
When you run Lumberyard tools such as Lumberyard Editor, an EditContext and a
SerializeContext are provided. You can use the robust facilities in these contexts to expose your
ﬁelds to content creators.
The following code demonstrates basic edit context reﬂection:
AZ::EditContext* edit = serialize->GetEditContext();
if (edit)
{
edit->Class<TestAsset>("My Component", "The World's Most Clever Component")
->ClassElement(AZ::Edit::ClassElements::EditorData, "")
->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu, AZ_CRC("Game"))
->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someFloatField, "Some
Float", "This is a float that means X.")
->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someStringField, "Some
String", "This is a string that means Y.")
->DataElement(AZ::Edit::UIHandlers::ComboBox, &MyComponent::m_someEnumField,
"Choose an Enum", "Pick an option among a set of enum values.")
->EnumAttribute(MyComponent::SomeEnum::EnumValue1, "Value 1")
->EnumAttribute(MyComponent::SomeEnum::EnumValue2, "Value 2")
->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_things, "Bunch of
Things", "A list of things for doing Z.")
;
}

Although this example demonstrates the simplest usage, many features and options are available when
you reﬂect structures (including components) to the edit context. For the ﬁelds to be exposed directly to
content creators, the example provides a friendly name and a description (tooltip) as the third and fourth
parameters of DataElement. For three ﬁelds, the ﬁrst parameter of DataElement is the default UI
handler AZ::Edit::UIHandlers::Default. The property system's architecture supports the ability
to add any number of UI handlers, each valid for one or more ﬁeld types. A given type can have multiple
available handlers, with one handler designated as the default. For example, ﬂoats by default use the
SpinBox handler, but a Slider handler is also available.
An example of binding a ﬂoat to a slider follows:
->DataElement(AZ::Edit::UIHandlers::Slider, &MyComponent::m_someFloatField, "Some Float",
"This is a float that means X.")
->Attribute(AZ::Edit::Attributes::Min, 0.f)
->Attribute(AZ::Edit::Attributes::Max, 10.f)
->Attribute(AZ::Edit::Attributes::Step, 0.1f)

Version 1.28
980


Reflecting a Component for Serialization and Editing

The `AZ::Edit::UIHandlers::Slider` UI handler expects `AZ::Edit::Attributes::Min` and `AZ::Edit::Attributes::Max` attributes. Optionally, you can provide a value for `AZ::Edit::Attributes::Step`. The example provides incremental increases of 0.1. If you do not provide a value for `AZ::Edit::Attributes::Step`, a default stepping of 1.0 is used.

**Note**

The property system supports external UI handlers, so you can implement your own UI handlers in your own modules. You can customize the behavior of the field, the Qt control that it uses, and the attributes that it observes.

For C++ API reference documentation on the edit context, see the `EditContext Class Reference` in the Amazon Lumberyard C++ API Reference.

**Attributes**

The example also demonstrates the use of attributes. Attributes are a generic construct on the edit context that allows the binding of literals, or functions that return values, to a named attribute. UI handlers can retrieve this data and use it to drive their functionality.

Attribute values can be bound to the following:

- **Literal values**
  
  ```
  Attribute(AZ::Edit::Attributes::Min, 0.0f)
  ```

- **Static or global variables**
  
  ```
  Attribute(AZ::Edit::Attributes::Min, &g_globalMin)
  ```

- **Member variables**
  
  ```
  Attribute(AZ::Edit::Attributes::Min, &MyComponent::m_min)
  ```

- **Static or global functions**
  
  ```
  Attribute(AZ::Edit::Attributes::ChangeNotify, &SomeGlobalFunction)
  ```

- **Member functions**
  
  ```
  Attribute(AZ::Edit::Attributes::ChangeNotify, &MyComponent::SomeMemberFunction)
  ```

**Change Notification Callbacks**

Another commonly used feature of the edit context is its ability to bind a change notification callback:

```c++
->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someStringField, "Some String", "This is a string that means Y."
->Attribute(AZ::Edit::Attributes::ChangeNotify, &MyComponent::OnStringFieldChanged)
```

The example binds a member function to be invoked when this property is changed, which allows the component to conduct other logic. The `AZ::Edit::Attributes::ChangeNotify` attribute also looks for an optional returned value that tells the property system if it needs to refresh aspects of its state. For example, if your change callback modifies other internal data that affects the property system, you can request a value refresh. If your callback modifies data that requires attributes be reevaluated (and any bound functions be reinvoked), you can request a refresh of attributes and values. Finally, if your callback conducts work that requires a full refresh (this is not typical), you can refresh the entire state.

The following example causes the property grid to refresh values when `m_someStringField` is modified through the property grid.

```c++
AZ::Edit::PropertyRefreshLevels::ValuesOnly signals the property grid to update the GUI with changes to the underlying data.
```
Reflecting a Component for Serialization and Editing

->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_someStringField, "Some String", "This is a string that means Y.")
->Attribute(AZ::Edit::Attributes::ChangeNotify, &MyComponent::OnStringFieldChanged)

AZ::u32 MyComponent::OnStringFieldChanged()
{
    m_someFloatField = 10.0f;

    // We've internally changed displayed data, so tell the property grid to refresh values (cheap).
    return AZ::Edit::PropertyRefreshLevels::ValuesOnly;
}

AZ::Edit::PropertyRefreshLevels::ValuesOnly is one of the following refresh modes that you can use (%dev\Code\Framework\AzCore\AzCore\Serialization\EditContextConstants.inl):

- **AttributesAndValues** – Reevaluates attributes of the properties displayed in the UI and refreshes their values. Because attributes can be bound to data members, member functions, global functions, or static variables, it's sometimes necessary to ask the property grid to reevaluate them. Doing so might include reinvoking bound functions.
- **EntireTree** – Refreshes the entire tree that is displayed in the UI.
- **None** – Specifies that the properties that are displayed in the UI should not be refreshed.
- **ValuesOnly** – Refreshes only the values of the properties that are displayed in the UI. The property grid updates the GUI to reflect changes to underlying data that might have occurred in the change callback.

The following more complex example binds a list of strings as options for a combo box. The list of strings is attached to a string field Property A. Suppose you want to modify the options available in the combo box for Property A with the values from another Property B. In that case you can bind the combo box AZ::Edit::Attributes::StringList attribute to a member function that computes and returns the list of options. In the AZ::Edit::Attributes::ChangeNotify attribute for Property B, you tell the system to reevaluate attributes, which in turn reinvokes the function that computes the list of options.

... bool m_enableAdvancedOptions;
AZStd::string m_useOption;
...

->DataElement(AZ::Edit::UIHandlers::Default, &MyComponent::m_enableAdvancedOptions, "Enable Advanced Options", "If set, advanced options will be shown.")
->Attribute(AZ::Edit::Attributes::ChangeNotify, AZ::Edit::PropertyRefreshLevels::AttributesAndValues)
->DataElement(AZ::Edit::Edit::UIHandlers::ComboBox, &MyComponent::m_useOption, "Options", "Available options.")
->Attribute(AZ::Edit::Attributes::StringList, &MyComponent::GetEnabledOptions)
...

AZStd::vector<AZStd::string> MyComponent::GetEnabledOptions()
{
    AZStd::vector<AZStd::string> options;
    options.reserve(16);
    options.push_back("Basic option");
    options.push_back("Another basic option");
Defining and Using Component Services

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Components can optionally specify a list of services that they provide, are incompatible with, depend on, or require. When you create a component, you can use this service specification to define relationships between various components. The component entity system uses this list for the conditional addition and removal of components at both edit time and at run time. The service specification also defines the order in which components are activated when an entity is activated. Specifically, components that provide services that another component depends on are activated first.

The following example shows a service specification.

```cpp
static void GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided)
{
    provided.push_back(AZ_CRC("ProvidedService"));
    provided.push_back(AZ_CRC("AnotherProvidedService"));
}
static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required)
{
    required.push_back(AZ_CRC("RequiredService"));
    required.push_back(AZ_CRC("AnotherRequiredService"));
}
static void GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible)
{
    incompatible.push_back(AZ_CRC("IncompatibleService"));
}
static void GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent)
{
    dependent.push_back(AZ_CRC("DependentOnService"));
}
```

**ProvidedService** – Specifies the service that the component provides. For example, a TransformComponent could provide a TransformService that in turn provides locational information.

**RequiredService** – Specifies a service that the component requires. The components that provide the required services are guaranteed to be present and active before this component is activated. For example, an audio component might need to know where it is located and therefore require a TransformService. Because of this requirement, the audio component can be added only to entities that have the component that provides the TransformService.

**DependentService** – Specifies a service on which the component depends but does not require. The component entity system guarantees that the components that provide dependent services are activated
before the component itself is activated. For example, an audio component could depend on the physics component. If the entity has physics, the audio component can query the physics component for physical material information. However, the audio component does not require that physics be present.

**IncompatibleService** – Specifies a service that cannot work with the component. Consider these examples:

- An entity can have only one type of collider. Therefore, the `PrimitiveColliderService` specifies that the `MeshColliderService` is incompatible with it and vice versa.
- The same effect can be achieved if two collider components already provide the `ColliderService` themselves and therefore specify the `ColliderService` as incompatible. Marking a component as incompatible with `ColliderService` ensures that no other component that has the same service is added to the entity.
- The `IncompatibleService` specification is frequently used to specify that multiples of the same component cannot exist on an entity.

## Editor Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Some components in Lumberyard have separate editor and runtime versions. The editor version is active in the editor. The runtime version is used for running the level in game or in the editor by pressing `Ctrl+G` or clicking `AI/Physics` below the viewport. Lumberyard uses editor components to maintain a clean separation between tools-specific code and data on one hand, and leaner runtime component data on the other. In general, runtime game components do not require editor counterparts. Components rarely need to be fully active at edit time. The light and mesh components are exceptions because they must behave the same at edit time as at run time.

*EditContext* reflection is fully supported in runtime components. Edit time is the only time when editor components are active. At run time, when Lumberyard processes a level or dynamic slice, it uses the runtime equivalents of editor components. Using the *EditContext* from a runtime component is usually sufficient to provide a rich editing experience.

**Important**

Editor components are not required. An editor component is necessary only if one of the following is true:

- Your component must be fully active at edit time. Edit time refers to standard editing mode; runtime components are used for the `AI/Physics` mode and gameplay (`Ctrl+G`).
- You must add special tools functionality to your component that requires that you compile only into your editor binaries.
- Your component provides functionality only in the editor and does not export a runtime component (for example, if your component manages selection logic).

## Sample Editor Component

The following code shows a sample editor component.

```c
/* Include the following headers: 
* EditorComponentBase.h - the editor component base class. Derive from 
* this class to create a component to use in the editor that is the 
* counterpart of the version of the component that is used during runtime. 
*/
```
Editor Component and Runtime Component Differences

The code for editor components is similar to the code for runtime components. The following sections list the key differences. It is safe to assume that editor component code is the same as it is for runtime component code other than the differences listed. For more information, see Creating a Component (p. 974).

Base Classes

All editor components include the AzToolsFramework::Components::EditorComponentBase class somewhere in their inheritance ancestry. If a component must display edit-time visualization, it must be a handler on the AzFramework::EntityDebugDisplayEventBus::Handler bus, as in the following example.

```cpp
#include <AzToolsFramework/ToolsComponents/EditorComponentBase.h>
#include <AzToolsFramework/ToolsComponents/EditorVisibilityBus.h>
#include <AzFramework/Entity/EntityDebugDisplayBus.h>
#include <MyComponent.h>

class MyEditorComponent
    : public AzToolsFramework::Components::EditorComponentBase
    , private AzFramework::EntityDebugDisplayEventBus::Handler
{
    public:
        AZ_EDITOR_COMPONENT(MyEditorComponent, "{5034A7F3-63DB-4298-83AA-915AB23EFEA0}");
        // Perform reflection for this component. The context parameter is the reflection context.
        static void Reflect(AZ::ReflectContext* context);

        // AZ::Component interface implementation.
        void Init() override;
        void Activate() override;
        void Deactivate() override;

        // AzFramework::EntityDebugDisplayEventBus implementation.
        void DisplayEntityViewport(const AzFramework::ViewportInfo& viewportInfo,
                                   AzFramework::DebugDisplayRequests& debugDisplay) override;

        // Optional functions for defining provided and dependent services.
        static void GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided);
        static void GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent);
        static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required);
        static void GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible);

        // Facilitate the translation of an editor component into a runtime component.
        void BuildGameEntity(AZ::Entity* gameEntity) override;
};
```
### Macro

Every editor component must specify the `AZ_EDITOR_COMPONENT` macro within its class definition. The macro takes two arguments:

1. The component type name.
2. A unique UUID. You may use any UUID generator to produce the value. Visual Studio provides this functionality through `Tools, Create GUID`. Use the `Registry Format` setting, and then copy and paste the value that is generated.

A sample `AZ_EDITOR_COMPONENT` macro follows.

```cpp
AZ_EDITOR_COMPONENT(MyEditorComponent, "{5034A7F3-63DB-4298-83AA-915AB23EFEA0}");
```

**Note**

Some Lumberyard editor components specify `AzToolsFramework::Components::EditorComponentBase` as the base class but use the `AZ_COMPONENT` instead of the `AZ_EDITOR_COMPONENT` macro, as in the following example.

```cpp
AZ_COMPONENT(EditorMannequinComponent, "{C5E08FE6-E1FC-4080-A053-2C65A667FE82}", AzToolsFramework::Components::EditorComponentBase);
```

### The DisplayEntityViewport Method

To draw debug visuals in the viewport for a specific entity, implement the `DisplayEntityViewport` method of the `AzFramework::EntityDebugDisplayEventBus` interface. Use this location for custom primitive edit-time visualization code.

```cpp
#include <AzFramework/Entity/EntityDebugDisplayBus.h>
...
void DisplayEntityViewport(const AzFramework::ViewportInfo& viewportInfo, AzFramework::DebugDisplayRequests& debugDisplay) override;
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewportInfo</td>
<td>Determines information such as camera position.</td>
</tr>
<tr>
<td>debugDisplay</td>
<td>Contains the interface for debug draw or display commands.</td>
</tr>
</tbody>
</table>

### The BuildGameEntity Method

The `BuildGameEntity` method from `EditorComponentBase.h` facilitates the translation of an editor component into a runtime component. Override this method as follows.

```cpp
#include <AzToolsFramework/ToolsComponents/EditorComponentBase.h>
...
void BuildGameEntity(AZ::Entity* gameEntity) override;
```

A typical implementation of the `BuildGameEntity` method performs the following actions:
Creating System Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

System components are similar to other components in Lumberyard's component entity framework. However, instead of creating game entity behavior, they control the behavior of the engine itself. System components are first-class elements of the game engine and are included at a deep level early in the initialization process. For more information, see System Components (p. 1098) in the AZ Modules (p. 1090) section.

Like any Lumberyard component (p. 974), a system component can provide services and can depend on or require other system component services. Lumberyard provides precise control over the order of engine initialization and system dependencies.

When you author system components, follow the best practices for component authoring. For example, your system components should use the following:

- Working with the Event Bus (EBus) system (p. 1851) to expose their interfaces.
- Reflection (p. 977) to serialize and edit settings in the Advanced Settings dialog of the Project Configurator (p. 1104).
- The same AZ::Component Functions (p. 976) for activation and deactivation.

Important
Just like game components, system components often provide request and notification buses. However, because system components are global systems, they should not specify IDs for their buses like game components. Game developers should be able to call your system's EBuses without having to deal with or know about the system entity that contains all system components.

Topics
- Creating a System Component in a Gem (p. 987)
- Making a Component a System Component (p. 989)

Creating a System Component in a Gem

Lumberyard enables the creation of custom system components through gems and AZ modules. Gems are a specialization of AZ modules. For more information, see Gems and AZ Modules (p. 1100). Most Lumberyard games organize their game code in one or more gems. These gems can contain system components that integrate with the game engine as well as components for use on game entities.

When you create a system component as part of a gem, note the following requirements:

- Your gem's GetRequiredSystemComponents() function must return the system component.
Creating System Components

- Your `GemNameBus.h` file goes under the `lumberyard_version\dev\Gems\GemName\Code\Include\GemName` directory.
- Your component source files go under the `lumberyard_version\dev\Gems\GemName\Code\Source` directory.

Example: HttpRequestorSystemComponent

The HttpRequestor gem (p. 1144) makes asynchronous HTTP/HTTPS requests and returns data through a callback function that you provide. The HttpRequestor gem uses the EBus for communication and provides all requests asynchronously. The HttpRequestorSystemComponent is part of the HttpRequestor gem. You can find the source code files for the gem under the `lumberyard_version\dev\Gems\HttpRequestor\Code` directory.

The HttpRequestor EBus

The following code shows the EBus implementation for HttpRequestor (`HttpRequestorBus.h`).

```cpp
#include <AzCore/EBus/EBus.h>
#include "HttpTypes.h"

namespace HttpRequestor
{
    class HttpRequestorRequests : public AZ::EBusTraits
    {
        public:
            static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;
            static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;

        // Public functions
            virtual void AddRequest(const AZStd::string& URI, Aws::Http::HttpMethod method, const Callback& callback) = 0;
            virtual void AddRequestWithHeaders(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const Callback& callback) = 0;
            virtual void AddRequestWithHeadersAndBody(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const AZStd::string& body, const Callback& callback) = 0;
            virtual void AddTextRequest(const AZStd::string& URI, Aws::Http::HttpMethod method, const TextCallback& callback) = 0;
            virtual void AddTextRequestWithHeaders(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const TextCallback& callback) = 0;
            virtual void AddTextRequestWithHeadersAndBody(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const AZStd::string& body, const TextCallback& callback) = 0;
        }
    using HttpRequestorRequestBus = AZ::EBus<HttpRequestorRequests>;
} // namespace HttpRequestor
```

The HttpRequestorSystemComponent Class

The following code shows the HttpRequestorSystemComponent class (`HttpRequestorSystemComponent.h`).

```cpp
#include <AzCore/Component/Component.h>
#include <HttpRequestor/HttpRequestorBus.h>
#include "HttpRequestManager.h"
```

Version 1.28
988
namespace HttpRequestor

class HttpRequestorSystemComponent : public AZ::Component,
  protected HttpRequestorRequestBus::Handler
{
  public:
    AZ_COMPONENT(HttpRequestorSystemComponent, "{CF29468F-1F67-497F-B4FF-C0F123584864}");

    static void Reflect(AZ::ReflectContext* context);
    static void GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided);
    static void GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible);
    static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required);
    static void GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent);

    protected:

    //////////////////////////////////////////////////////////////////////////
    // HttpRequestorRequestBus interface implementation
    //////////////////////////////////////////////////////////////////////////
    void AddRequest(const AZStd::string& URI, Aws::Http::HttpMethod method, const Callback& callback) override;
    void AddRequestWithHeaders(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const Callback& callback) override;
    void AddRequestWithHeadersAndBody(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const AZStd::string& body, const Callback& callback) override;
    void AddTextRequest(const AZStd::string& URI, Aws::Http::HttpMethod method, const TextCallback& callback) override;
    void AddTextRequestWithHeaders(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const TextCallback& callback) override;
    void AddTextRequestWithHeadersAndBody(const AZStd::string& URI, Aws::Http::HttpMethod method, const Headers & headers, const AZStd::string& body, const TextCallback& callback) override;

    //////////////////////////////////////////////////////////////////////////
    // AZ::Component interface implementation
    //////////////////////////////////////////////////////////////////////////
    void Init() override;
    void Activate() override;
    void Deactivate() override;

    //////////////////////////////////////////////////////////////////////////
    private:
    ManagerPtr m_httpManager;
};

Making a Component a System Component

After you create the code for your component, add it to your project’s system entity to make it a system component.

**To define a component as a system component**

1. Use the `GetRequiredSystemComponents()` function to add your component to the system entity for your project during application startup.
The following example is from HttpRequestorModule.cpp.

```cpp
#include "HttpRequestor_precompiled.h"
#include "HttpRequestorSystemComponent.h"
#include <AzCore/Module/Module.h>

namespace HttpRequestor
{
    class HttpRequestorModule
    : public AZ::Module
    {
        public:
            AZ_RTTI(HttpRequestorModule, "{FD411E40-AF83-4F6B-A5A3-F59AB71150BF}",
                     AZ::Module);

            HttpRequestorModule()
            : AZ::Module()
            {
                // Push results of [MyComponent]:CreateDescriptor() into m_descriptors here.
                m_descriptors.insert(m_descriptors.end(), {
                    HttpRequestorSystemComponent::CreateDescriptor(),
                });
            }

            /**
             * Add required SystemComponents to the SystemEntity.
             */
            AZ::ComponentTypeList GetRequiredSystemComponents() const override
            {
                return AZ::ComponentTypeList{
                    azrtti_typeid<HttpRequestorSystemComponent>(),
                };
            }
    }
}
```

2. (Optional) Expose the system component to the System Entity Editor (p. 50). This enables game developers to configure the component’s properties on a per-project basis. To do so, reflect the system component to the EditContext and set the AppearsInAddComponentMenu field to System.

The following example is from HttpRequestorSystemComponent.cpp.

```cpp
#include "HttpRequestor_precompiled.h"
#include <AzCore/Serialization/SerializeContext.h>
#include <AzCore/Serialization/EditContext.h>
#include "HttpRequestorSystemComponent.h"

namespace HttpRequestor
{
    void HttpRequestorSystemComponent::Reflect(AZ::ReflectContext* context)
    {
        if (AZ::SerializeContext* serialize = azrtti_cast<AZ::SerializeContext*>(context))
        {
            serialize->Class<HttpRequestorSystemComponent>();
        }
    }
```
The "HttpRequestor" and "Will make HTTP Rest calls" string parameters specify the UI name and tooltip information for the component in the Add Component list. The Category field specifies the group in which the component appears. In this case, no category is specified, so the group is Miscellaneous by default.

The following image shows the result in the System Entity Editor.
For detailed steps on using the System Entity Editor to add a system component to a project, see Configuring System Entities (p. 1104).

**Programmer's Guide to Component Mode**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
If you're a programmer, you can edit your components to add support for Component Mode. This can help designers on your team work more easily with components. You can use this feature to do the following:

- Use manipulators to edit a component directly in the viewport
- Create editing modes and tools without needing to create custom standalone editors
- Create your own shortcuts

For more information, see Editing Components in the Viewport (p. 483).

**Topics**

- Step 1: Choose a Component (p. 993)
- Step 2: Add a ComponentModeDelegate (p. 994)
- Step 3: Write a New Component Mode (p. 1000)
- Step 4: Add Shortcuts for Component Mode (p. 1009)
- Step 5: Handle Selection in the Viewport (p. 1011)
- Step 6: Building Component Mode (p. 1013)
- Tips for Working with Component Mode (p. 1014)

**Note**
Before you start, you should be familiar with the component entity framework and have written a component. For more information, see the Programmer's Guide to Entities and Components (p. 973).

**Step 1: Choose a Component**

Select a component to add a Component Mode to. Components such as shape, light, and physics are good examples because their data exists in the 3D world and you can view the changes directly in the viewport. Other components such as Audio and Script Canvas are less useful because they have properties that you can't edit in the viewport.

This walkthrough uses the Point Light (p. 706) component because it's visually easy to model. At the end of the walkthrough, you can edit the Point Light component in the viewport instead of manually entering values in the Entity Inspector.

**Note**
Component Mode is an edit-time only feature. For more information, see Editor Components (p. 984).

Before

By default, you can't edit the Point Light component in the viewport. You can edit its properties only in the Entity Inspector.
After you add a Component Mode to a component, you can click and drag the manipulator to adjust the Point Light component's area of effect.

For a list of available components, see Component Reference (p. 532).

Step 2: Add a ComponentModeDelegate

To add this feature to the Point Light component, you must update the component so that it can detect the intent to begin editing a component and manage the creation of a Component Mode. To do so, add a new type called a ComponentModeDelegate to your existing component.

To add a ComponentModeDelegate

1. Navigate to the EditorPointLightComponent.h file and open the file in a text editor.
2. Add the ComponentModeDelegate.
Example EditorPointLightComponent.h

Before

```
#pragma once

#include "EditorLightComponent.h"

namespace LmbrCentral
{
    /*
    * In-editor Point Light component.
    * Handles previewing and activating lights in the editor.
    */
    class EditorPointLightComponent
    : public EditorLightComponent
    {
        public:
            AZ_COMPONENT(EditorPointLightComponent, "{00818135-138D-42AD-8657-FF3FD38D9E7A}", AzToolsFramework::Components::EditorComponentBase);

            static void Reflect(AZ::ReflectContext* context);

            void Init() override;

        protected:
            const char* GetLightTypeText() const override
            {
                return "Point Light";
            }
    } // class EditorPointLightComponent
} // namespace LmbrCentral
```

After

```
See the following code changes.

```
#pragma once

#include "EditorLightComponent.h"

// BEGIN ADD
#include <AzToolsFramework/ComponentMode/ComponentModeDelegate.h>
// END ADD

namespace LmbrCentral
{
    /*
    * In-editor Point Light component.
    * Handles previewing and activating lights in the editor.
    */
    class EditorPointLightComponent
    : public EditorLightComponent
    {
        public:
            AZ_COMPONENT(EditorPointLightComponent, "{00818135-138D-42AD-8657-FF3FD38D9E7A}", AzToolsFramework::Components::EditorComponentBase);

            static void Reflect(AZ::ReflectContext* context);

            void Init() override;
    } // class EditorPointLightComponent
} // namespace LmbrCentral
```
In this example, there's an #include for ComponentModeDelegate.h, which is where you add the file as a member. The using declaration reduces the length of the m_componentDelegate definition. The updated file also provides Activate and Deactivate member functions for the component.

3. Save the file.

Now that you've updated the EditorPointLightComponent.h file, you must make changes to the EditorPointLightComponent.cpp implementation file.

To update the EditorPointLightComponent.cpp

1. In a text editor, open the EditorPointLightComponent.cpp file.
2. Add an #include for the unwritten Component Mode and comment out the change. This becomes the EditorPointLightComponentMode.h file.

   // #include "EditorPointLightComponentMode.h"

3. Serialize the m_componentModeDelegate member to expose it to the EditContext.

   ->Field("ComponentMode", &EditorPointLightComponent::m_componentModeDelegate)

4. Enter the following changes so that the ComponentModeDelegate appears in the Entity Inspector. This change adds the Edit button to the Point Light component.

   ->DataElement(AZ::Edit::UIHandlers::Default, &EditorPointLightComponent::m_componentModeDelegate, "Component Mode", "Point Light Component Mode")
   ->Attribute(AZ::Edit::Attributes::Visibility, AZ::Edit::PropertyVisibility::ShowChildrenOnly)

   **Tip**

   The Attribute ensures that the Edit button appears inline without being nested inside the ComponentModeDelegate class/struct.

5. Inside the Activate call, add the following code.
m_componentModeDelegate.ConnectWithSingleComponentMode(
    EditorPointLightComponent, /*EditorPointLightComponentMode*/(
        AZ::EntityComponentIdPair(GetEntityId(), GetId()), nullptr);

a. The helper function ConnectWithSingleComponentMode handles the most common case of adding a Component Mode. This call wraps a more complex API where multiple Component Modes can be activated at the same time.

For example, see the EditorTubeShapeComponent.h file.

b. Specify the component type and Component Mode itself.

   Note
   The template parameters generalize connecting and instantiating Component Modes. For more information, see the CreateComponentModeBuilder.h and its AddComponentModes function.

c. Specify the EntityId that the component is attached to and the ComponentId.

   Note
   It’s possible (although not often used) to address an EBus not only by the EntityId but by the EntityId and ComponentId pair. A ComponentId isn’t guaranteed to be unique on its own, but when it’s combined with an EntityId, you can target a specific component attached to an entity. This is useful when you have multiple components of the same type attached to a single entity.

d. The final argument is a nullptr, which is an EditorComponentSelectionRequestBus::Handler. To use this handler, the EditorComponent must implement the EditorComponentSelectionRequestsBus. If the EBus was implemented, you can pass ‘this’ here, but as you haven’t implemented it yet, enter nullptr for now.

   These changes enable you to double-click in the viewport to select a component. You implement the EditorComponentSelectionRequestBus in Step 5: Handle Selection in the Viewport (p. 1011).

   For examples, see the EditorBaseShapeComponent.h and EditorSplineComponent.h files.

6. Enter the following code to disconnect the ComponentModeDelegate when the EditorComponent is deactivated.

m_componentModeDelegate.Disconnect();

This change ensures that the ComponentModeDelegate disconnects from the various EBuses that are connected to in the Activate function.

Example EditorPointLightComponent.cpp

Your code should look like the following.

Before

```cpp
#include "LmbrCentral_precompiled.h"
#include "EditorPointLightComponent.h"
#include <AzCore/Serialization/SerializeContext.h>
#include <AzCore/Serialization/EditContext.h>
#include <AzCore/RTTI/BehaviorContext.h>
```
namespace LmbrCentral
{
    void EditorPointLightComponent::Reflect(AZ::ReflectContext* context)
    {
        if (auto serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
        {
            serializeContext->Class<EditorPointLightComponent, EditorLightComponent>()
                ->Version(1);
        }

        if (AZ::EditContext* editContext = serializeContext->GetEditContext())
        {
            editContext->Class<EditorPointLightComponent>(
                "Point Light", "The Point Light component allows an entity to create a point of light")
                ->ClassElement(AZ::Edit::ClassElements::EditorData,"
                    ->Attribute(AZ::Edit::Attributes::Category, "Rendering")
                    ->Attribute(AZ::Edit::Attributes::Icon, "Editor/Icons/Components/PointLight.png")
                    ->Attribute(AZ::Edit::Attributes::ViewportIcon, "Editor/Icons/Components/Viewport/PointLight.png")
                    ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                        AZ_CRC("Game", 0x232b318c))
                    ->Attribute(AZ::Edit::Attributes::AutoExpand, true)
                    ->Attribute(AZ::Edit::Attributes::Visibility,
                        AZ::Edit::PropertyVisibility::ShowChildrenOnly)
                    ->Attribute(AZ::Edit::Attributes::HelpPageURL, "https://docs.aws.amazon.com/lumberyard/latest/" userguide/component-point-light.html")
                ;
        }
    }

    if (auto behaviorContext = azrtti_cast<AZ::BehaviorContext*>(context))
    {
        behaviorContext->Class<EditorPointLightComponent>()
            ->RequestBus("EditorPointLightComponentBus");
    }

    void EditorPointLightComponent::Init()
    {
        SetLightType(EditorLightConfiguration::LightType::Point);
        EditorLightComponent::Init();
    }
} // namespace LmbrCentral

After

#include "LmbrCentral_precompiled.h"
#include "EditorPointLightComponent.h"

// BEGIN ADD
// #include "EditorPointLightComponentMode.h"
// END ADD

#include <AzCore/Serialization/SerializeContext.h>
#include <AzCore/Serialization/EditContext.h>
#include <AzCore/RTTI/BehaviorContext.h>

namespace LmbrCentral
{
    void EditorPointLightComponent::Reflect(AZ::ReflectContext* context)
    {
        if (auto serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
        {
            serializeContext->Class<EditorPointLightComponent, EditorLightComponent>()
                ->Version(1);
        }

        if (AZ::EditContext* editContext = serializeContext->GetEditContext())
        {
            editContext->Class<EditorPointLightComponent>(
                "Point Light", "The Point Light component allows an entity to create a point of light")
                ->ClassElement(AZ::Edit::ClassElements::EditorData,"
                    ->Attribute(AZ::Edit::Attributes::Category, "Rendering")
                    ->Attribute(AZ::Edit::Attributes::Icon, "Editor/Icons/Components/PointLight.png")
                    ->Attribute(AZ::Edit::Attributes::ViewportIcon, "Editor/Icons/Components/Viewport/PointLight.png")
                    ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                        AZ_CRC("Game", 0x232b318c))
                    ->Attribute(AZ::Edit::Attributes::AutoExpand, true)
                    ->Attribute(AZ::Edit::Attributes::Visibility,
                        AZ::Edit::PropertyVisibility::ShowChildrenOnly)
                    ->Attribute(AZ::Edit::Attributes::HelpPageURL, "https://docs.aws.amazon.com/lumberyard/latest/" userguide/component-point-light.html")
                ;
        }
    }

    if (auto behaviorContext = azrtti_cast<AZ::BehaviorContext*>(context))
    {
        behaviorContext->Class<EditorPointLightComponent>()
            ->RequestBus("EditorPointLightComponentBus");
    }

    void EditorPointLightComponent::Init()
    {
        SetLightType(EditorLightConfiguration::LightType::Point);
        EditorLightComponent::Init();
    }
} // namespace LmbrCentral

Version 1.28
998
->Version(1)
  // BEGIN ADD
  ->Field("ComponentMode",
    &EditorPointLightComponent::m_componentModeDelegate)
  // END ADD

if (AZ::EditContext* editContext = serializeContext->GetEditContext())
{
  editContext->Class<EditorPointLightComponent>(
      "Point Light", "The Point Light component allows an entity to
      create a point of light")
    ->ClassElement(AZ::Edit::ClassElements::EditorData, "")
    ->Attribute(AZ::Edit::Attributes::Category, "Rendering")
    ->Attribute(AZ::Edit::Attributes::Icon, "Editor/Icons/
      Components/PointLight.png")
    ->Attribute(AZ::Edit::Attributes::ViewportIcon, "Editor/Icons/
      Components/Viewport/PointLight.png")
    ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
      AZ_CRC("Game", 0x232b318c))
    ->Attribute(AZ::Edit::Attributes::AutoExpand, true)
    ->Attribute(AZ::Edit::Attributes::Visibility,
      AZ::Edit::PropertyVisibility::ShowChildrenOnly)
    ->Attribute(AZ::Edit::Attributes::HelpPageURL, "https://
      docs.aws.amazon.com/lumberyard/latest/userguide/component-point-light.html")
  // BEGIN ADD
  ->DataElement(AZ::Edit::UIHandlers::Default,
    &EditorPointLightComponent::m_componentModeDelegate, "Component Mode", "Point Light
    Component Mode")
  ->Attribute(AZ::Edit::Attributes::Visibility,
    AZ::Edit::PropertyVisibility::ShowChildrenOnly)
  // END ADD
}

if (auto behaviorContext = azrtti_cast<AZ::BehaviorContext*>(context))
{
  behaviorContext->Class<EditorPointLightComponent>()-
    >RequestBus("EditorPointLightComponentBus");
}

void EditorPointLightComponent::Init()
{
  SetLightType(EditorLightConfiguration::LightType::Point);
  EditorLightComponent::Init();
}

// BEGIN ADD
void EditorPointLightComponent::Activate()
{
  EditorLightComponent::Activate();
  // m_componentModeDelegate.ConnectWithSingleComponentMode<
    // EditorPointLightComponent, EditorPointLightComponentMode>(
      AZ::EntityComponentIdPair(GetEntityId(), GetId()), nullptr);
}

void EditorPointLightComponent::Deactivate()
{
  // m_componentModeDelegate.Disconnect();
  EditorLightComponent::Deactivate();
}
// END ADD
Step 3: Write a New Component Mode

Now that you've updated the header and implementation file, this procedure shows you how to write a new Component Mode. To do so, you create a EditorPointLightComponentMode.h file, which is the interface for Component Mode.

To write a new Component Mode

1. Navigate to the \lumberyard_version\dev\Gems\LmbrCentral\Code\Source\Rendering directory.
2. In a text editor, create a file and name it EditorPointLightComponentMode.h.
3. Enter the following code. This includes the EditorBaseComponentMode.h file so that you can inherit from the EditorBaseComponentMode.

   ```cpp
   #include <AzToolsFramework/ComponentMode/EditorBaseComponentMode.h>
   ```

4. Ensure that all Component Modes must inherit from EditorBaseComponentMode.

   ```cpp
   : public AzToolsFramework::ComponentModeFramework::EditorBaseComponentMode
   ```

5. For the EditorBaseComponentMode, enter the following code to override the Refresh function.

   ```cpp
   // EditorBaseComponentMode
   void Refresh() override;
   ```

   The Refresh function is called after any undo or redo action to ensure that a Component Mode is updated to reflect the current state of the component. For example, the manipulator positions must be updated after you undo or redo an action.

   **Note**
   The EditorBaseComponentMode interface is designed to be as light as possible. Opt-in only the parts that you need.

6. Declare the manipulator to modify a specific property on the component. The example uses the LinearManipulator, so that you can adjust a point along a given axis.

   ```cpp
   AZStd::shared_ptr<AzToolsFramework::LinearManipulator>
   m_pointMaxDistanceManipulator; ///< Manipulator for point max distance property.
   ```

   **Note**
   • You must use an AZStd::shared_ptr to manage the lifetime of the LinearManipulator property, which the ManipulatorManager requires.
   • In this procedure, you use only the LinearManipulator property, but there are other properties available. The PlanarManipulator allows two degrees of freedom to edit a value, and the AngularManipulator can rotate a value. You can also use aggregate manipulators such as the TranslationManipulators and RotationManipulators. You can also create and extend your own manipulators by inheriting from BaseManipulator. However, this is an advanced topic and isn't encouraged because you can achieve most functionality by customizing behavior in the existing manipulator callbacks.

7. Save your file.

   **Example EditorPointLightComponentMode.h**

   Your code should look like the following. Note the light interface.
#pragma once

#include <AzToolsFramework/ComponentMode/EditorBaseComponentMode.h>

namespace AzToolsFramework
{
    class LinearManipulator;
}

namespace LmbrCentral
{
    class EditorPointLightComponentMode
    : public AzToolsFramework::ComponentModeFramework::EditorBaseComponentMode
    {
        public:
            EditorPointLightComponentMode(
                const AZ::EntityComponentIdPair& entityComponentIdPair, AZ::Uuid componentType);
            ~EditorPointLightComponentMode();

            // EditorBaseComponentMode
            void Refresh() override;

        private:
            AZStd::shared_ptr<AzToolsFramework::LinearManipulator> m_pointMaxDistanceManipulator; /// Manipulator for point max distance property.
        };
    } // namespace LmbrCentral

Implement Component Mode

Now that you've written the interface portion of a Component Mode, create a EditorPointLightComponentMode.cpp file. This file implements a Component Mode for the Point Light component.

In this procedure, you make the following changes to the file:

1. Construction (p. 1003)
2. Manipulator Setup (p. 1003)
3. Manipulator Callbacks (p. 1006)

Example EditorPointLightComponentMode.cpp

After you complete this procedure, your EditorPointLightComponentMode.cpp file looks like the following.

#include "LmbrCentral_precompiled.h"
#include "EditorPointLightComponentMode.h"

#include <AzCore/Component/TransformBus.h>
#include <AzToolsFramework/Manipulators/LinearManipulator.h>
#include <AzToolsFramework/Manipulators/ManipulatorManager.h>

#include <LmbrCentral/Rendering/EditorLightComponentBus.h>

namespace LmbrCentral
{
    EditorPointLightComponentMode::EditorPointLightComponentMode(
        const AZ::EntityComponentIdPair& entityComponentIdPair, AZ::Uuid componentType)
: EditorBaseComponentMode(entityComponentIdPair, componentType)
{
    AZ::Transform worldFromLocal = AZ::Transform::CreateIdentity();
    AZ::TransformBus::EventResult(
        worldFromLocal, GetEntityId(), &AZ::TransformInterface::GetWorldTM);

    m_pointMaxDistanceManipulator = AzToolsFramework::LinearManipulator::MakeShared(worldFromLocal);
    m_pointMaxDistanceManipulator->AddEntityId(GetEntityId());
    m_pointMaxDistanceManipulator->SetAxis(AZ::Vector3::CreateAxisX());
    Refresh();

    const AZ::Color manipulatorColor(0.3f, 0.3f, 0.3f, 1.0f);
    const float manipulatorSize = 0.05f;

    AzToolsFramework::ManipulatorViews views;
    views.emplace_back(AzToolsFramework::CreateManipulatorViewQuadBillboard(manipulatorColor, manipulatorSize));
    m_pointMaxDistanceManipulator->SetViews(AZStd::move(views));

    struct SharedState
    {
        float m_startingPointMaxDistance = 0.0f;
    };

    auto sharedState = AZStd::make_shared<SharedState>();
    m_pointMaxDistanceManipulator->InstallLeftMouseDownCallback(
        [this, sharedState] (const AzToolsFramework::LinearManipulator::Action& action) mutable
            {
                float currentMaxDistance = 0.0f;
                EditorLightComponentRequestBus::EventResult(
                    currentMaxDistance, GetEntityId(), &EditorLightComponentRequests::GetPointMaxDistance);
                sharedState->m_startingPointMaxDistance = currentMaxDistance;
            });

    m_pointMaxDistanceManipulator->InstallMouseMoveCallback(
        [this, sharedState](const AzToolsFramework::LinearManipulator::Action& action)
            {
                const AZ::VectorFloat axisDisplacement = action.LocalPositionOffset().Dot(action.m_fixed.m_axis);
                EditorLightComponentRequestBus::Event(
                    GetEntityId(), &EditorLightComponentRequests::SetPointMaxDistance,
                    (sharedState->m_startingPointMaxDistance + axisDisplacement).GetMax(AZ::VectorFloat(0.1f)));

                const AZ::Vector3 localPosition = action.LocalPosition().GetMax(AZ::Vector3(0.1f, 0.0f, 0.0f));
                m_pointMaxDistanceManipulator->SetLocalTransform(AZ::Transform::CreateTranslation(localPosition));
                m_pointMaxDistanceManipulator->SetBoundsDirty();
                // ensure property grid values are refreshed
                AzToolsFramework::ToolsApplicationNotificationBus::Broadcast(
                    &AzToolsFramework::ToolsApplicationNotificationBus::Events::InvalidatePropertyDisplay,
                    AzToolsFramework::Refresh_Values);
            });

    m_pointMaxDistanceManipulator->Register(AzToolsFramework::g_mainManipulatorManagerId);
```cpp
EditorPointLightComponentMode::~EditorPointLightComponentMode()
{
    m_pointMaxDistanceManipulator->Unregister();
}

void EditorPointLightComponentMode::Refresh()
{
    float currentMaxDistance = 0.0f;
    EditorLightComponentRequestBus::EventResult(
        currentMaxDistance, GetEntityId(),
        &EditorLightComponentRequests::GetPointMaxDistance);
    m_pointMaxDistanceManipulator->SetLocalTransform(
        AZ::Transform::CreateTranslation(AZ::Vector3::CreateAxisX() *
        currentMaxDistance));
} // namespace LmbrCentral
```

**Note**

Most of the code in the file is related to the manipulator. Manipulators are low-level but provide a large degree of control.

**Construction**

The Component Mode constructor contains the majority of the logic.

**To construct a Component Mode**

1. Navigate to the `lumberyard_version\dev\Gems\LmbrCentral\Code\Source\Rendering` directory.
2. In a text editor, create a file and name it `EditorPointLightComponentMode.cpp`.
3. Call the `EditorBaseComponentMode` constructor and specify the `Entity` and `ComponentId`, along with the `componentType`.

```cpp
EditorPointLightComponentMode::EditorPointLightComponentMode(
    const AZ::EntityComponentIdPair& entityComponentIdPair, AZ::Uuid componentType)
    : EditorBaseComponentMode(entityComponentIdPair, componentType) // IMPORTANT
```

**Manipulator Setup**

Next, set up the manipulator for the component.

**To set up the manipulator**

1. In the `EditorPointLightComponentMode.cpp` file, identify the `LinearManipulator` in the code.
2. Request the world transform of the entity that the component is attached to and pass that value to the constructor of the manipulator. You must set the space that the manipulator is going to operate in. With components, this value is usually the entity transform. If you want the manipulator to operate in world space, you can pass the identity transform here.

```cpp
AZ::Transform worldFromLocal = AZ::Transform::CreateIdentity();
AZ::TransformBus::EventResult(
    worldFromLocal, GetEntityId(), &AZ::TransformInterface::GetWorldTM);
```
Note

The naming `worldFromLocal` is chosen to indicate how this transform is modifying a position. For example, if you have a position in the local space of the entity, this transform takes it from local to world space. The naming style helps debug the multiplication order of transforms and vectors. Lumberyard uses column major ordering, which is a matrix multiplication that occurs right to left.

For example, if you have the vector, `localPosition` and the transform `worldFromLocal`, multiplying `worldFromLocal * localPosition` has the correct output because the local identifiers are next to each other. This transforms the `localPosition` to its position in world space.

3. (Optional) Add the `EntityId` to the manipulator. This is helpful to track manipulator undo and redo operations on entities.

During each mouse move, the added `EntityIds` are marked as `dirty`. When a manipulator action ends, Lumberyard compares the entity and component serialized state before and after the event. If the entity changed, Lumberyard records an undo step. If not, Lumberyard throws away the potential undo action. It's important to note that this tracks the change the manipulator caused on the serialized entity state. If you have other custom operations that you want to undo, create a new `UndoCommand` that derives from `URSequencePoint`.

```cpp
m_pointMaxDistanceManipulator->AddEntityId(GetEntityId());
```

4. For the `SetAxis` function, specify a vector in the local space of the entity. This defines the vector that the `LinearManipulator` moves along in local space. The following example uses the x-axis, but you can specify another vector.

```cpp
m_pointMaxDistanceManipulator->SetAxis(AZ::Vector3::CreateAxisX());
```

5. To set the position of the manipulator, query the `EditorLightComponent`. You don't have a direct reference (pointer) to the component or entity. All communication is made using EBuses.

```cpp
// From void EditorPointLightComponentMode::Refresh()
float currentMaxDistance = 0.0f;
EditorLightComponentRequestBus::EventResult(
    currentMaxDistance, GetEntityId(),
    &EditorLightComponentRequests::GetPointMaxDistance);
```

Note

Using EBuses and EntityIds offers the following advantages:

- You don't need to couple a Component Mode to a specific component. For example, `Box Shape` and `PhysX Collider` components need similar editing capabilities, such as the ability to resize the dimensions of an oriented bounding box in the viewport. The `EditorBoxShapeComponent.h` and `EditorPhysXColliderComponent.h` files include the `BoxManipulatorRequestBus`. This provides an interface to get the shape or collider transform and get or set its dimensions. This way, you can apply the `BoxComponentMode` for both files.

- You can avoid difficulties with entities being destroyed and recreated with each undo and redo action. If an entity changes while recording an undo action, the act of undoing the action destroys the current entity and recreates it by returning the entity to its previous saved state. If a Component Mode has a direct reference to the `EditorComponent` and not just an `EntityId`, managing the lifetimes would be more complex. This means that the `EditorComponent` that you want to edit must expose the get and set actions.
that you need on its request bus. Otherwise, a Component Mode can't read or write the actions.

6. The call to `SetLocalTransform` sets the transform of the manipulator. By default, this value is the same local space of the entity. To calculate this value, query the current `PointMaxDistance` and offset the manipulator handle by that distance along the x-axis.

In the following example, the `ManipulatorView` doesn't have an orientation. You can specify `CreateTranslation` on the `Transform` class.

```cpp
// From void EditorPointLightComponentMode::Refresh()
m_pointMaxDistanceManipulator->SetLocalTransform(AZ::Transform::CreateTranslation(AZ::Vector3::CreateAxisX() * currentMaxDistance));
```

7. Configure the `ManipulatorView`.

The behavior of a manipulator is decoupled from how it appears in the viewport. This means that a `LinearManipulator` can look like a line, cone, cube, or screen-aligned quad. `LinearManipulator` supports multiple views, which is useful with the classic `TranslationManipulator`. You can draw a line and cone (an arrow) to represent the `LinearManipulator` that corresponds to each axis.

8. Create an `AZStd::vector` of views and a `QuadBillboardView` that specifies the color and dimensions of the shape.

9. Add the new views to the manipulator itself with `SetViews`.

**Example**

```cpp
const AZ::Color manipulatorColor(0.3f, 0.3f, 0.3f, 1.0f);
const float manipulatorSize = 0.05f;
AZToolsFramework::ManipulatorViews views;
views.emplace_back(AZToolsFramework::CreateManipulatorViewQuadBillboard(manipulatorColor, manipulatorSize));
m_pointMaxDistanceManipulator->SetViews(AZStd::move(views));
```

10. Save the file.

**Example**

Your code should look like the following so far.

```cpp
// EditorPointLightComponentMode::EditorPointLightComponentMode()
AZ::Transform worldFromLocal = AZ::Transform::CreateIdentity();
AZ::TransformBus::EventResult(
    worldFromLocal, GetEntityId(), &AZ::TransformInterface::GetWorldTM);

m_pointMaxDistanceManipulator =
    AZToolsFramework::LinearManipulator::MakeShared(worldFromLocal);
m_pointMaxDistanceManipulator->AddEntityId(GetEntityId());
m_pointMaxDistanceManipulator->SetAxis(AZ::Vector3::CreateAxisX());

// Refresh(); inlined/expanded
float currentMaxDistance = 0.0f;
EditorLightComponentRequestBus::EventResult(
    currentMaxDistance, GetEntityId(), &EditorLightComponentRequests::GetPointMaxDistance);

m_pointMaxDistanceManipulator->SetLocalTransform(AZ::Transform::CreateTranslation(AZ::Vector3::CreateAxisX() * currentMaxDistance));
const AZ::Color manipulatorColor(0.3f, 0.3f, 0.3f, 1.0f);
```
const float manipulatorSize = 0.05f;

AzToolsFramework::ManipulatorViews views;
views.emplace_back(AzToolsFramework::CreateManipulatorViewQuadBillboard(manipulatorColor,
manipulatorSize));

m_pointMaxDistanceManipulator->SetViews(AZStd::move(views));

### Manipulator Callbacks

Next, set up how the manipulator should respond when you interact with it in the viewport.

### To set up manipulator callbacks

1. In the EditorPointLightComponentMode.cpp file, enter the following code to create a piece of shared state that each callback can use.

```cpp
struct SharedState
{
    float m_startingPointMaxDistance = 0.0f;
};

auto sharedState = AZStd::make_shared<SharedState>();
```

**Note**

You can add a member to the EditorPointLightComponentMode and refer to that in each lambda expression. However, because only the lambda expressions care about this state, keep its scope as constrained as possible.

2. Use AZStd::shared_ptr to ensure that the lambda expressions capture the pointer by value. This guarantees that the lambda expressions own the shared state and effectively close over it. This is similar to a closure in JavaScript.

```cpp
m_pointMaxDistanceManipulator->InstallLeftMouseDownCallback(
    [this, sharedState] (const AzToolsFramework::LinearManipulator::Action& /*action*/) mutable
    {
        float currentMaxDistance = 0.0f;
        EditorLightComponentRequestBus::EventResult(
            currentMaxDistance, GetEntityId(),
            &EditorLightComponentRequests::GetPointMaxDistance);

        sharedState->m_startingPointMaxDistance = currentMaxDistance;
    });
```

3. Reference the data contained in EditorPointLightComponentMode so that it can capture the this pointer.

```cpp
m_pointMaxDistanceManipulator->InstallMouseMoveCallback(
    [this, sharedState] (const AzToolsFramework::LinearManipulator::Action& action)
    {
        const AZ::VectorFloat axisDisplacement =
            action.LocalPositionOffset().Dot(action.m_fixed.m_axis);

        EditorLightComponentRequestBus::Event(
            GetEntityId(), &EditorLightComponentRequests::SetPointMaxDistance,
            (sharedState->m_startingPointMaxDistance +
            axisDisplacement).GetMax(AZ::VectorFloat(0.1f)));

        const AZ::Vector3 localPosition = action.LocalPosition().GetMax(AZ::Vector3(0.1f, 0.0f, 0.0f));
```
Lumberyard User Guide
Programmer's Guide to Component Mode

m_pointMaxDistanceManipulator->SetLocalTransform(AZ::Transform::CreateTranslation(localPosition));

m_pointMaxDistanceManipulator->SetBoundsDirty();

// ensure property grid values are refreshed
AzToolsFramework::ToolsApplicationNotificationBus::Broadcast(
    &AzToolsFramework::ToolsApplicationNotificationBus::Events::InvalidatePropertyDisplay,
    AzToolsFramework::Refresh_Values);
});

The callback that you get from the LinearManipulator passes a struct called Action. This struct contains information about the current state of the manipulator.

4. To determine how far you move along the axis, add a .Dot between the LocalPositionOffset and the fixed axis of the manipulator. This gives you a projection of the LocalPositionOffset onto the axis and the distance moved.

```cpp
const AZ::VectorFloat axisDisplacement = action.LocalPositionOffset().Dot(action.m_fixed.m_axis);
```

The manipulator action has three parts. You can specify these properties to control how they should modify your component.

- **Fixed** – Contains data associated with the manipulator, which is set at creation. Often, this is the axis or plane of movement.
- **Start** – State of the manipulator at MouseDown.
- **Current** – Current state of the manipulator during a MouseMove.

5. After you calculate the axisDisplacement, specify that value to set the current PointMaxDistance, which updates the state of the component. You must update the LocalTransform of the manipulator. If you don't, the visual representation of the manipulator doesn't change. To do this, read the LocalPosition of the action and set the LocalTransform of the manipulator.

```cpp
const AZ::Vector3 localPosition = action.LocalPosition().GetMax(AZ::Vector3(0.1f, 0.0f, 0.0f));
m_pointMaxDistanceManipulator->SetLocalTransform(AZ::Transform::CreateTranslation(localPosition));
```

6. After you update the LocalTransform of the manipulator, you must mark its bounds as dirty so that they can be recalculated for intersection tests.

```cpp
m_pointMaxDistanceManipulator->SetBoundsDirty();
```

7. Enter the following code to notify the editor that you modified a property that needs to be refreshed in the Entity Inspector. If you don't make this update, the component's properties in the Entity Inspector don't match with how the component appears in the viewport.

```cpp
AzToolsFramework::ToolsApplicationNotificationBus::Broadcast(
    &AzToolsFramework::ToolsApplicationNotificationBus::Events::InvalidatePropertyDisplay,
    AzToolsFramework::Refresh_Values);
```

8. Register the manipulator with the main ManipulatorManager. This ensures that this manipulator is associated with the main viewport.

```cpp
m_pointMaxDistanceManipulator->Register(AzToolsFramework::g_mainManipulatorManagerId);
```
9. You must unregister the manipulator from the ManipulatorManager when it gets destroyed. To do this, add this function to your Component Mode destructor.

```cpp
EditorPointLightComponentMode::~EditorPointLightComponentMode()
{
    m_pointMaxDistanceManipulator->Unregister();
}
```

**Example**

In the following code, override `InstallLeftMouseDownCallback` and `InstallMouseMoveCallback` callbacks to achieve the preferred behavior. This code lists the callback logic.

```cpp
struct SharedState
{
    float m_startingPointMaxDistance = 0.0f;
};
auto sharedState = AZStd::make_shared<SharedState>();

m_pointMaxDistanceManipulator->InstallLeftMouseDownCallback(
    [this, sharedState](const AzToolsFramework::LinearManipulator::Action& /*action*/)
    mutable
    {
        float currentMaxDistance = 0.0f;
        EditorLightComponentRequestBus::EventResult(
            currentMaxDistance, GetEntityId(),
            &EditorLightComponentRequests::GetPointMaxDistance);
        sharedState->m_startingPointMaxDistance = currentMaxDistance;
    });

m_pointMaxDistanceManipulator->InstallMouseMoveCallback(
    [this, sharedState](const AzToolsFramework::LinearManipulator::Action& action)
    {
        const AZ::VectorFloat axisDisplacement = action.LocalPositionOffset().Dot(action.m_fixed.m_axis);
        EditorLightComponentRequestBus::Event(
            GetEntityId(), &EditorLightComponentRequests::SetPointMaxDistance,
            (sharedState->m_startingPointMaxDistance +
                axisDisplacement).GetMax(AZ::VectorFloat(0.1f)));

        const AZ::Vector3 localPosition = action.LocalPosition().GetMax(AZ::Vector3(AZ::VectorFloat(0.1f)));
        m_pointMaxDistanceManipulator->SetLocalTransform(AZ::Transform::CreateTranslation(localPosition));
        m_pointMaxDistanceManipulator->SetBoundsDirty();
        // ensure property grid values are refreshed
        AzToolsFramework::ToolsApplicationNotificationBus::Broadcast(
            &AzToolsFramework::ToolsApplicationNotificationBus::Events::InvalidatePropertyDisplay,
            AzToolsFramework::Refresh_Values);
    });
```

10. To see a Component Mode in action, switch back to the `EditorPointLightComponent.h` and remove the comment lines that you added to your code. See `EditorPointLightComponent.cpp` (p. 995).

```cpp
#include "EditorPointLightComponentMode"
```
Step 4: Add Shortcuts for Component Mode

In the following procedure, create shortcuts that are specific to the component that you're editing. For example, if you're editing a Spline component and want to add shortcuts for working with vertices, you can create a shortcut such as Ctrl+D to duplicate a point or Delete to remove it. When adding shortcuts, note the following changes in your code:

- EditorBaseComponentMode provides an additional function to override in its interface called PopulateActionsImpl.

```cpp
AZStd::vector<AzToolsFramework::ActionOverride> PopulateActionsImpl() override;
```

- The PopulateActionsImpl function returns a vector of available actions that you can perform in Component Mode. By default, you always get the following shortcuts, which you can't change:
  - Ctrl+S – Save
  - Ctrl+Z – Undo
  - Ctrl+Shift+Z – Redo
  - Esc – Leave Component Mode

To create shortcuts for Component Mode

1. In the EditorPointLightComponentMode.cpp file, create an ActionOverride struct and use the Set chain methods to improve readability. The most important piece of this change is a unique identifier. The format is a reverse URL: com.company.action.component.type.

```cpp
static const AZ::Crc32 s_resetPointLight = AZ_CRC("com.amazon.action.pointlight.reset") ;
```

2. Specify what shortcut to use and then enter a name and description so that the shortcut appears in the Edit menu.

```cpp
.SetKeySequence(QKeySequence(Qt::Key_R))
.SetTitle("Reset")
.SetTip("Reset all Point Light State")
.SetEntityComponentIdPair(AZ::EntityComponentIdPair(GetEntityId(), GetComponentId()));
```

Note
You must provide the entity and component ID that this action corresponds to and finally the event or callback itself to perform.

3. To ensure that you can undo this action, create an undo batch and mark the entity as dirty.

```cpp
// Ensure that we record undo command for reset
AzToolsFramework::ScopedUndoBatch undoBatch("Reset Point Light");
AzToolsFramework::ScopedUndoBatch::MarkEntityDirty(entityId);
```
4. The current and next states of the entity are compared, and if they don't match, the undo action is recorded. Follow the `undoBatch` function with a series of calls to reset the state of the component and then refresh and update the position of any manipulators so that they accurately reflect the state of the component.

```cpp
/// Other set functions to return state to default...
// refresh manipulators
Refresh();

// ensure property grid values are refreshed
AzToolsFramework::ToolsApplicationNotificationBus::Broadcast(
    &AzToolsFramework::ToolsApplicationNotificationBus::Events::InvalidatePropertyDisplay,
    AzToolsFramework::Refresh_Values);
```

5. Save your file.

**Example EditorPointLightComponentMode.cpp**

You can add additional actions such as a shortcut to reset all properties on the component like the following.

```cpp
static const AZ::Crc32 s_resetPointLight = AZ_CRC("com.amazon.action.pointlight.reset") ;
AZStd::vector<AzToolsFramework::ActionOverride>
EditorPointLightComponentMode::PopulateActionsImpl()
{
    return AZStd::vector<AzToolsFramework::ActionOverride>
    {
        AzToolsFramework::ActionOverride()
        .SetUri(s_resetPointLight)
        .SetKeySequence(QKeySequence(Qt::Key_R))
        .SetTitle("Reset")
        .SetTip("Reset all Point Light State")
        .SetEntityComponentIdPair(AZ::EntityComponentIdPair(GetEntityId(), GetComponentId()))
        .SetCallback([this]() {
            const AZ::EntityId entityId = GetEntityId();

            // ensure we record undo command for reset
            AzToolsFramework::ScopedUndoBatch undoBatch("Reset Point Light");
            AzToolsFramework::ScopedUndoBatch::MarkEntityDirty(entityId);

            EditorLightComponentRequestBus::Event(
                entityId, &EditorLightComponentRequests::SetPointMaxDistance, 1.0f);

            /// other set functions to return state to default...

            // refresh manipulators
            Refresh();

            // ensure property grid values are refreshed
            AzToolsFramework::ToolsApplicationNotificationBus::Broadcast(
                &AzToolsFramework::ToolsApplicationNotificationBus::Events::InvalidatePropertyDisplay,
                AzToolsFramework::Refresh_Values);
        });
    }
};
```
Step 5: Handle Selection in the Viewport

In the following procedure, make changes to your code so that you can enter Component Mode by double-clicking the component in the viewport.

In Component Mode, you can modify the dimensions of the **Point Light** component directly in the viewport.

**To handle selection in the viewport**

1. In a text editor, open the `EditorPointLightComponent.h` file.
2. For the last parameter, add the `EditorComponentSelectionRequestsBus::Handler`.

```cpp
class EditorPointLightComponent
    : public EditorLightComponent,
     private AzToolsFramework::EditorComponentSelectionRequestsBus::Handler
```

3. To implement the `EditorComponentSelectionRequests`, you must override the following four functions:
   1. `GetEditorSelectionBoundsViewport` – Returns an AABB encompassing the visible extents of your component
   2. `EditorSelectionIntersectRayViewport` – Where you implement selection for the component
   3. `SupportsEditorRayIntersect` – Override this function and return true if you implemented `EditorSelectionIntersectRayViewport`
   4. `GetBoundingBoxDisplayType` – Used for debugging to ensure that the AABB is the correct fit.
      This example sets the function to `NoBoundingBox`

```cpp
// EditorComponentSelectionRequests
AZ::Aabb GetEditorSelectionBoundsViewport(const AzFramework::ViewportInfo& viewportInfo) override;
bool EditorSelectionIntersectRayViewport(const AzFramework::ViewportInfo& viewportInfo, const AZ::Vector3& src, const AZ::Vector3& dir, AZ::VectorFloat& distance) override;
bool SupportsEditorRayIntersect() override;
AZ::u32 GetBoundingBoxDisplayType() override;
```

4. Save the file.
5. In a text editor, open the `EditorPointLightComponent.cpp` file.
6. Connect and disconnect from the `EditorComponentSelectionRequestsBus` in the `Activate` and `Deactivate` functions of the component.

```cpp
void EditorPointLightComponent::Activate()
{
    ...
    AzToolsFramework::EditorComponentSelectionRequestsBus::Handler::BusConnect(GetEntityId());
    ...
}

void EditorPointLightComponent::Deactivate()
{
    ...
    AzToolsFramework::EditorComponentSelectionRequestsBus::Handler::BusDisconnect();
    ...
```
7. Add the following changes to your code:

- Add an implementation of `SupportsEditorRayIntersect` to return `true`. By default, this function returns `false`.
- Add an implementation of `GetBoundingBoxDisplayType` to return:
  
  ```cpp
  AzToolsFramework::EditorComponentSelectionRequests::BoundingBoxDisplay::NoBoundingBox
  ```

```cpp
bool EditorPointLightComponent::SupportsEditorRayIntersect()
{
    return true;
}

AZ::u32 EditorPointLightComponent::GetBoundingBoxDisplayType()
{
    return AzToolsFramework::EditorComponentSelectionRequests::BoundingBoxDisplay::NoBoundingBox;
}
```

**Note**

It's possible to instead return the:

```cpp
AzToolsFramework::EditorComponentSelectionRequests::BoundingBoxDisplay::BoundingBox
```

for debugging, but you shouldn't leave it enabled.

The next two functions show how to implement the picking and selection support.

8. Add the implementation for the `GetEditorSelectionBoundsViewport` function.

9. Create an AABB centered around the component covering its extents. In this case, get the position in world space of the entity and create an AABB with the radius of the point light. Because the point light is represented as a sphere, use the `GetPointMaxDistance` function.

**Example**

Your code should look like the following.

```cpp
AZ::Aabb EditorPointLightComponent::GetEditorSelectionBoundsViewport(
    const AzFramework::ViewportInfo& viewportInfo)
{
    AZ::Vector3 worldTranslation = AZ::Vector3::CreateZero();
    AZ::TransformBus::EventResult(
        worldTranslation, GetEntityId(), &AZ::TransformInterface::GetWorldTranslation);
    return AZ::Aabb::CreateCenterRadius(worldTranslation, GetPointMaxDistance());
}
```

In the next step, make changes to the `EditorSelectionIntersectRayViewport` function.

**Example**

```cpp
// top of file
<AzToolsFramework/Picking/Manipulators/ManipulatorBounds.h>

...

bool EditorPointLightComponent::EditorSelectionIntersectRayViewport(
    const AzFramework::ViewportInfo& viewportInfo,
    const AZ::Vector3& src, const AZ::Vector3& dir, AZ::VectorFloat& distance)
{
```
AZ::Transform worldFromLocal = AZ::Transform::CreateIdentity();
AZ::TransformBus::EventResult(worldFromLocal, GetEntityId(), &AZ::TransformInterface::GetWorldTM);

const float minorRadius = 0.1f;
const float majorRadius = GetPointMaxDistance();

const AZ::Vector3 axes[] = {
    AZ::Vector3::CreateAxisX(), AZ::Vector3::CreateAxisY(),
    AZ::Vector3::CreateAxisZ()
};

enum { AxisCount = 3 };
float distances[AxisCount] = { FLT_MAX, FLT_MAX, FLT_MAX };
bool intersection = false;
for (size_t axisIndex = 0; axisIndex < AxisCount; ++axisIndex)
{
    intersection = intersection
        || AzToolsFramework::Picking::IntersectHollowCylinder(
            src, dir, worldFromLocal.GetTranslation(), axes[axisIndex],
            minorRadius, majorRadius, distances[axisIndex]);
}

return intersection;

10. Get the position of the entity in world space and approximate a torus or flat hollow cylinder to
represent the rings of the Point Light component. The minor radius corresponds to the tube part of
the torus, which is its thickness.

11. You want a radius that is a reasonable size so that you can easily select it in the viewport. The major
radius is the distance from the center of the torus to the middle of the tube. Because you have a
ring for each axis, check that each one is using the IntersectHollowCylinder function, which
basically approximates a torus.

12. Test a ring for each axis and store the intersection distances to find the closest intersection.

    { 
        intersection = intersection
            || AzToolsFramework::Picking::IntersectHollowCylinder(
                src, dir, worldFromLocal.GetTranslation(), axes[axisIndex],
                minorRadius, majorRadius, distances[axisIndex]);
    }

    distance = AZ::GetMin(AZ::GetMin(distances[0], distances[1]), distances[2]);

13. Save the file.

Step 6: Building Component Mode

In the following procedure, you must build the editor so that you can view your changes in Lumberyard
Editor.

To build Component Mode

1. Navigate to the lumberyard_version\dev\Gems\LmbrCentral\Code directory.
2. In a text editor, open the `lmbrcentral_editor.waf_files.waf` file.
3. Navigate to the "Source/Rendering": section and add the files that you created for a Component Mode.

**Example**

```
"Source/Rendering/EditorPointLightComponent.h",
"Source/Rendering/EditorPointLightComponent.cpp",
"Source/Rendering/EditorPointLightComponentMode.h",
"Source/Rendering/EditorPointLightComponentMode.cpp",
```

4. Save the file.
5. Navigate to the `lumberyard_version/`dev directory.
6. In a command-line window, enter the following commands to build the editor.

```
lmbr_waf configure

lmbr_waf build_win_x64_vs2017_profile -p all
```

7. After your build succeeds, open Lumberyard Editor and choose or create a level.
8. In the viewport, create an entity, attach the **Point Light** component, and do one of the following:

   - Double-click the entity
   - In the **Entity Inspector**, choose **Edit**

9. Make changes to your component in the viewport. You can select the manipulator on the **Point Light** component to adjust the radius of the light area.
10. When you've finished, choose **Done**.

**Tips for Working with Component Mode**

See the following tips for adding a Component Mode to your component.

**Additional Input Handling**

If you want to handle mouse events in a specific way (for example, outside of using manipulators), you can override `HandleMouseInteraction`, which is part of the `EditorBaseComponentMode` class that is defined in the `ComponentModeRequestBus`.

---

**Version 1.28**

1014
This `HandleMouseInteraction` function is called whenever a mouse event happens. The `MouseInteractionEvent`, which is passed as an argument to the function, contains information such as the mouse pick ray, position in screen space, and buttons and modifier keys that are pressed.

```cpp
// EditorBaseComponentMode
void Refresh() override;
bool HandleMouseInteraction(
    const AzToolsFramework::ViewportInteraction::MouseInteractionEvent& mouseInteraction)
override;
```

**Additional Drawing**

If you want to add additional debug drawing while in Component Mode, you can implement the `AzFramework::EntityDebugDisplayEventBus` and then override `DisplayEntityViewport`. Remember to connect the `EntityDebugDisplayEventBus` in the Component Mode constructor and disconnect in the destructor.

This can be useful for additional drawing for an entity and its components.

**Note**

The `EntityDebugDisplayEventBus` is only addressed by `EntityId`. You might encounter issues when working with multiple components of the same type on a given entity.

See the following example for the `.h` file.

```cpp
// EditorPointLightComponentMode.h
// AzFramework::EntityDebugDisplayEventBus
void DisplayEntityViewport(
    const AzFramework::ViewportInfo& viewportInfo,
    AzFramework::DebugDisplayRequests& debugDisplay) override;
```

See the following example for the implementation `.cpp` file.

```cpp
// EditorPointLightComponentMode.cpp
EditorPointLightComponentMode::EditorPointLightComponentMode(
    const AZ::EntityComponentIdPair& entityComponentIdPair, AZ::Uuid componentType)
: EditorBaseComponentMode(entityComponentIdPair, componentType)
{
    ... 
    AzFramework::EntityDebugDisplayEventBus::Handler::BusConnect(GetEntityId());
}

EditorPointLightComponentMode::~EditorPointLightComponentMode()
{
    AzFramework::EntityDebugDisplayEventBus::Handler::BusDisconnect();
    ...
}

void EditorPointLightComponentMode::DisplayEntityViewport(
    const AzFramework::ViewportInfo& /*viewportInfo*/,
    AzFramework::DebugDisplayRequests& debugDisplay)
{
    ... 
}
```

**Components and EBuses**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
EBuses are not required for components, nor are they directly bound to components in any way. But because they form the backbone of communication among all Lumberyard components, EBuses offer many benefits. We highly recommend that you learn how to use them in your game, systems, and components. For more information, see Working with the Event Bus (EBus) system (p. 1851).

Most components provide two EBuses to facilitate communication: a request bus and a notification bus. Both these EBuses use the EBusAddressPolicy::ById address policy and the ID of the entity for identification.

**Request Bus**

A component's request bus allows other components or external systems to make requests of the component. Usually the runtime version of the component implements the request bus. However, the editor component can service the bus in special cases.

The following sections examine the individual parts of an example request bus.

**Transform Request Event Group**

The following example defines a group of events that the TransformComponent handles.

```cpp
class TransformComponentRequests
 : public AZ::ComponentBus // EBus traits for component buses: identification is based on an entity ID.
{
    public:

    // EBusTraits overrides - Only a single handler is allowed for a given entity ID.
    // Only one component on a entity can implement the events.
    static const EBusHandlerPolicy HandlerPolicy = EBusHandlerPolicy::Single;

    // Returns the local transform (parent transform excluded).
    virtual const Transform& GetLocalTM() = 0;

    // Sets the local transform and notifies all interested parties.
    virtual void SetLocalTM(const Transform& /*tm*/) {}

    // Returns the world transform (including parent transform).
    virtual const Transform& GetWorldTM() = 0;

    // Sets the world transform and notifies all interested parties.
    virtual void SetWorldTM(const Transform& /*tm*/) {}

    // Returns both local and world transforms.
    virtual void GetLocalAndWorld(Transform& /*localTM*/, Transform& /*worldTM*/) {}

    ...;
};
```

**Base Class and Trait Specification**

The base class for most AZ::Component request buses is AZ::ComponentBus. This class is a convenience to help set up EBus traits typical of component EBuses. You could also set up EBus traits by inheriting the default AZ::EBusTraits. Then you could optionally override any or all of the following traits. For more information, see EBus Configuration Options (p. 1854).

- Address policy
- Bus ID type
- Connection policy
- Handler policy
- Lock type
- Priority sorting

These two approaches are shown in the following examples.

```cpp
// Example using AZ::ComponentBus.
class TransformComponentRequests : public AZ::ComponentBus {
(...)
```

```cpp
// Example using AZ::EBusTraits
class TransformComponentRequests : public AZ::EBusTraits {
{
(...)
  // EBusTraits overrides.
  static const EBusAddressPolicy AddressPolicy = EBusAddressPolicy::ById; // OR YOUR CHosen POLICY
  static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Multiple; // OR YOUR CHosen POLICY
  using BusIdType = EntityId;
(...)
```

**EBus Request Bus Events**

EBus event definitions are the main part of the bus specification. This interface defines what your component does. In the following example, the TransformComponent allows the retrieval and modification of the local and world transforms. It also creates interfaces for setting parent–child relationships.

```cpp
...
// Returns the local transform (parent transform excluded).
virtual const Transform& GetLocalTM() = 0;

// Sets the local transform and notifies all interested parties.
virtual void SetLocalTM(const Transform& /*tm*/) {}

// Returns the world transform (including parent transform).
virtual const Transform& GetWorldTM() = 0;

// Sets the world transform and notifies all interested parties.
virtual void SetWorldTM(const Transform& /*tm*/) {}

// Returns both local and world transforms.
virtual void GetLocalAndWorld(Transform& /*localTM*/, Transform& /*worldTM*/) {};
...
```

**EBus Request Bus Definition**

After the event group has been declared, the EBus must be defined. Although you can use `AZ::EBus<TransformComponentRequests>` to define an EBus, we recommend that you use a typedef instead, as in the following example. This improves readability at bus call sites.

```cpp
typedef AZ::EBus<TransformComponentRequests> TransformComponentRequestBus;
```
Another best practice is to use descriptive names in EBuses and avoid overloaded functions. Explicit and descriptive function names prevent future API name collisions as classes inherit (potentially many of) your EBus interfaces. Avoiding overloaded functions improves the experience of using your EBuses from scripting environments. In Lua and in visual scripting, the extra expressiveness improves readability and clarity.

### Notification Bus

A component uses its notification bus to inform other components and the rest of the engine about relevant changes. To do this, it sends notifications in the form of EBus events to any class that monitors the bus. To monitor the bus, classes implement the notification bus handler interface (in the case of TransformComponent, this is `AZ::TransformNotificationBus::Handler`.)

**Note**

A request bus sends messages to a component; a notification bus sends messages from a component.

### Transform Notification Event Group

The following example defines a group of notification events that the TransformComponent sends.

```cpp
class TransformNotifications : public AZ::ComponentBus
{
public:
...
    // Called when the local transform of the entity has changed. Local transform update always implies world transform change too.
    virtual void OnTransformChanged(const Transform& /*local*/, const Transform& /*world*/) {}
};
```

The notification bus can also change its EBusTrait specification if required.

### Components as EBus Handlers

After you have created the EBus event groups and defined the EBuses, your component can implement the EBus interface by deriving the EBus handler. The following example is from the TransformComponent.

```cpp
class TransformComponent : public AZ::Component,
    private AZ::TransformComponentRequestBus::Handler
{
    ...

    // TransformBus.

    /// Returns true if the tm was set to the local transform.
    const AZ::Transform& GetLocalTM() override { return m_localTM; }

    /// Sets the local transform and notifies all interested parties.
    void SetLocalTM(const AZ::Transform& tm) override;

    /// Returns true if the transform was set to the world transform.
    const AZ::Transform& GetWorldTM() override { return m_worldTM; }
```
Tick Bus and Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The tick bus is the primary mechanism by which components subscribe to events that occur per simulation frame. Instead of connecting components to the tick bus, strive to make your components entirely driven by events. If your component requires tick-based functionality, it can implement the tick bus interface’s `OnTick` method and connect to the tick bus for the required period of time. To avoid poor scalability of polling-based update structures, components should limit the time that they are connected to the tick bus.

Traditionally, a component might use the `OnTick` method to check the state continuously but actively do processing for only a fraction of the connected time. Instead, the component should connect to the tick bus only when the state changes and disconnect after processing is complete. This approach is closer to that of event-based programming, in which polling is conducted only for short periods of time or not at all.

As an example, imagine a component that monitors the state of an entity after the entity enters a trigger. The component should avoid subscribing to the tick bus until the entity has entered the trigger. It should then disconnect from the tick bus as soon as the entity has left the trigger.

In the following example, the `NavigationComponent` implements the `OnTick` method.

```cpp
class NavigationComponent
    : public AZ::Component
    , public NavigationComponentRequestBus::Handler
    , public AZ::TickBus::Handler
{
    ... 
    // TickBus
    virtual void OnTick(float deltaTime, AZ::ScriptTimePoint time);
    ... 
}
```

To connect and disconnect from the tick bus, the component uses code like the following.

```cpp
AZ::TickBus::Handler::BusConnect();

AZ::TickBus::Handler::BusDisconnect();
```
Customizing Tick Order

By default, a handler receives events based on the order in which the components are initialized. To control the order that your component receives OnTick events, you can override the GetTickOrder() function to return a custom integer value. The integer value determines the order in which your component is ticked relative to other components on the tick bus. Lower values are ticked before higher values. Any value is permitted. For convenience, the AZ::ComponentTickBus enum (TickBus.h) provides some preset values. These values are shown in the following table.

### Tick Order Preset Values

<table>
<thead>
<tr>
<th>Name (C++)</th>
<th>Name (Lua/Script Canvas)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICK_FIRST</td>
<td>TickOrder.First</td>
<td>0</td>
<td>First position in the tick handler order.</td>
</tr>
<tr>
<td>TICK_PLACEMENT</td>
<td>TickOrder.Placement</td>
<td>50</td>
<td>Suggested tick handler position for components that need to be early in the tick order.</td>
</tr>
<tr>
<td>TICK_INPUT</td>
<td>TickOrder.Input</td>
<td>75</td>
<td>Suggested tick handler position for input components.</td>
</tr>
<tr>
<td>TICK_GAME</td>
<td>TickOrder.Game</td>
<td>80</td>
<td>Suggested tick handler position for game-related components.</td>
</tr>
<tr>
<td>TICK_ANIMATION</td>
<td>TickOrder.Animation</td>
<td>100</td>
<td>Suggested tick handler position for animation components.</td>
</tr>
<tr>
<td>TICK_PHYSICS</td>
<td>TickOrder.Physics</td>
<td>200</td>
<td>Suggested tick handler position for physics components.</td>
</tr>
<tr>
<td>TICK_ATTACHMENT</td>
<td>TickOrder.Attachment</td>
<td>500</td>
<td>Suggested tick handler position for attachment components.</td>
</tr>
<tr>
<td>TICK_PRE_RENDER</td>
<td>TickOrder.PreRender</td>
<td>750</td>
<td>Suggested tick handler position to update render-related data.</td>
</tr>
<tr>
<td>TICK_DEFAULT</td>
<td>TickOrder.Default</td>
<td>1000</td>
<td>Default tick handler position when the handler is constructed.</td>
</tr>
<tr>
<td>TICK_UI</td>
<td>TickOrder.UI</td>
<td>2000</td>
<td>Suggested tick handler position for UI components.</td>
</tr>
<tr>
<td>TICK_LAST</td>
<td>TickOrder.Last</td>
<td>10000</td>
<td>Last position in the tick handler order.</td>
</tr>
</tbody>
</table>

The following code examples show how to override the GetTickOrder() function in Lua and in C++.

```lua
-- Lua example
defunction MyLuaUIComponent:GetTickOrder()
    return TickOrder.UI
end
```

```cpp
// C++ example
int MyCppUIComponent::GetTickOrder()
{
    return TICK_UI;
}
```
Note
As of Lumberyard version 1.11, use of the `TickEvents::m_tickOrder` variable is deprecated. If you change the value of `m_tickOrder` instead of overriding `GetTickOrder()`, you will receive a warning. However, your component will still tick in the appropriate order.

Event-Based Programming and Event-Based Polling: Best Practices
It is important to know when to use the tick bus and when to use event-driven programming patterns instead.

Event-Based Polling
It is often convenient to tick a component every frame and monitor the state of other entities. For example, a `LookAt` camera component is commonly implemented to tick each frame, retrieve the transform of the target entity, and update its own transform accordingly.

Event-Based Programming
In Lumberyard, a more event-driven approach is to use the `TransformBus` to monitor the target entity for transform changes in a purely event-driven fashion. If the target entity doesn’t move, no work is done and no polling is required. When the target entity moves, the `LookAt` component adjusts its own entity's transform accordingly.

Use Notifications to Make Your Components Easy to Use
When authoring a component, try to anticipate the requirements of components that might depend on yours. Use a notification bus to expose the appropriate notifications for your component. This approach enables others to write code that consumes the services of your components in a faster and more scalable way.

For more best practices, see Components and EBuses: Best Practices (p. 1026).

Exposing Custom Components to Track View for Animation

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To include custom components in cinematic cut scenes and movies rendered to disk, you must expose animatable component properties to Lumberyard's Track View and `Entity Inspector`. To expose a custom component and its properties, you must perform three steps:

1. Create getter and setter methods for the animated property on one of the component’s request event buses.
2. Implement the getter and setter request handlers in your component.
3. Reflect your component to the edit context and the behavior context. Edit context reflection exposes your component in `Entity Inspector`, and behavior context reflection exposes it in the Track View.

Exposing a Custom Component: Example
The following example assumes that a custom component called `ImaginaryTargetComponent` has been created. The component has a `Vector3` property called `ImaginaryPosition` that you want...
to animate in Track View. A request bus called ImaginaryTargetComponentBus has also been created for the component. This example assumes that you are familiar with programming event buses and component handlers for them. For more information, see Working with the Event Bus (EBus) system (p. 1851) and Creating a Component (p. 974).

To expose a custom component to Track View

1. Create getter and setter methods

   Each property must provide a method to set its value and get its current value. To implement this, create setter and getter methods on one of the component's request buses. Then reflect those methods to the behavior context as part of the class reflection for the component.

   The following example creates setter and getter requests on the ImaginaryTargetComponentRequestBus.

   ```c++
   /*! *
   * ImaginaryTargetComponentRequests EBus Interface
   * Messages serviced by ImaginaryTargetComponents.
   */
   class ImaginaryTargetComponentRequests : public AZ::ComponentBus
   {
   public:

   // EBusTraits overrides - Application is a singleton.
   // Only one component on an entity can implement the events.
   static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;

   virtual AZ::Vector3 GetImaginaryTargetPosition() = 0;
   virtual void SetImaginaryTargetPosition(const AZ::Vector3& newPosition) = 0;
   
   using ImaginaryTargetComponentRequestBus = AZ::EBus<ImaginaryTargetComponentRequests>;
   }
   ```

2. Implement handlers in your component

   Implement handlers in your component for the setter and getter requests that you declared in the first step, as in the following example.

   ```c++
   class ImaginaryTargetComponent : public AzToolsFramework::Components::EditorComponentBase,
   public LmbrCentral::ImaginaryTargetComponentRequestBus::Handler
   {
   public:

   A2_EDITOR_COMPONENT(ImaginaryTargetComponent, "{4491D282-C120-4B2E-BC63-AC86296956A2}");

   ImaginaryTargetComponent() : m_imaginaryPosition(.0f) {};

   // ImaginaryTargetComponentRequestBus::Handler implementation.

   // Presumably these would be used for something useful; this example just
   // stores and returns the value.
   AZ::Vector3 GetImaginaryTargetPosition() override { return m_imaginaryPosition; }
   void SetImaginaryTargetPosition(const AZ::Vector3& newPosition) override
   { m_imaginaryPosition = newPosition; }

   protected:

   // Required Reflect function.
   static void Reflect(AZ::ReflectContext* context);
   ```
3. **Reflect your component**

Using the edit context and behavior contexts, reflect the component's class, request event bus, and setter and getter methods. Track View uses the setter and getter methods that you reflect in this step to set and get values for your animated property. You must also reflect a VirtualProperty declaration that tells Track View that your component is capable of being animated.

```cpp
/*static*/ void ImaginaryTargetComponent::Reflect(AZ::ReflectContext* context)
{
    AZ::SerializeContext* serializeContext =
        azrtti_cast<AZ::SerializeContext*>(context);

    if (serializeContext)
    {
        serializeContext->Class<ImaginaryTargetComponent,
            AzToolsFramework::Components::EditorComponentBase>()
            ->Version(0)
            ->Field("ImaginaryPosition",
                &ImaginaryTargetComponent::m_imaginaryPosition);

        AZ::EditContext* editContext = serializeContext->GetEditContext();
        if (editContext)
        {
            editContext->Class<ImaginaryTargetComponent>("ImaginaryTarget", "A Code Sample enabling Track View Animation")
                ->ClassElement(AZ::Edit::ClassElements::EditorData,"
                    ->Attribute(AZ::Edit::Attributes::Category, "Game")
                    ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                        AZ_CRC("Game", 0x232b318c))
                ->DataElement(0, &ImaginaryTargetComponent::m_imaginaryPosition,
                    "Imaginary Target Pos", "Imaginary Target Position")
            ;
        }
    }

    AZ::BehaviorContext* behaviorContext = azrtti_cast<AZ::BehaviorContext*>(context);
    if (behaviorContext)
    {
        // Reflect the setter and getter methods and create a virtual property that
        // refers to them.
        behaviorContext->EBus<ImaginaryTargetComponentRequestBus>("ImaginaryTargetRequestBus")
            ->Event("GetImaginaryTargetPosition",
                &ImaginaryTargetComponentRequestBus::Events::GetImaginaryTargetPosition)
            ->Event("SetImaginaryTargetPosition",
                &ImaginaryTargetComponentRequestBus::Events::SetImaginaryTargetPosition)
                ->VirtualProperty("ImaginaryPosition", "GetImaginaryTargetPosition",
                    "SetImaginaryTargetPosition");

        // Attach the "ImaginaryTargetRequestBus" EBus that you reflected to the
        // behavior context of the ImaginaryTargetComponent class.
        behaviorContext->Class<ImaginaryTargetComponent>()
            ->RequestBus("ImaginaryTargetRequestBus");
    }
}
```
4. (Optional) Place Unit Attributes on Getters

The Track View user interface depends on the data type that the getter and setter use. The foregoing example uses a type of AZ::Vector3, so Track View creates a compound x, y, z track from the property. By contrast, if the getter and setters use a bool, Track View creates a Boolean track. For the majority of animatable properties, the type is sufficient. However, in some cases you might have to set units for a reflected property. For example, if your property's AZ::Vector3 represents a color, you must add an attribute to the reflection of the getter event. The attribute instructs Track View to use a color picker for that property. If you have a property called ImaginaryTargetColor that calls a getter event called GetImaginaryTargetColor, use reflection code like the following:

```cpp
->Event("GetImaginaryTargetColor",
    &ImaginaryTargetComponentRequestBus::Events::GetImaginaryTargetColor)
->Attribute("Units", AZ::Edit::Attributes:: PropertyUnits8BitColor)
```

Track View then uses a color track for the property, as the following image shows.

Other units can be found in the file dev\Code\Framework\AZCore\AZCore\Serialization \EditContextConstants.inl. As of Lumberyard release 1.8, these units are the following.

```cpp
const static AZ::Crc32 PropertyUnitsRadian = AZ_CRC("Radians");
const static AZ::Crc32 PropertyUnits8BitColor = AZ_CRC("8BitColor");
```

If you have an angular parameter in radians that you want to Track View to convert to degrees in its user interface, use AZ::Crc32 PropertyUnitsRadian.

Viewing the Result

Now you can view how the example component and property appear in the Entity Inspector and the Track View.
In the following Entity Inspector image, EditContext reflection has exposed the ImaginaryTarget component and its Imaginary Target Pos property.

In the following Track View image, BehaviorContext reflection has exposed the ImaginaryTarget component and the ImaginaryPosition track from the corresponding virtual property.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Follow these best practices for creating and using components and EBuses.

**EBus Names**

The following EBus naming conventions remove ambiguity and provide consistency.

- Use the name format `MyComponentRequestBus` for the bus that others use to invoke functions on `MyComponent`, as in the following example.

  ```cpp
class CheeseburgerComponentRequests : public AZ::ComponentBus
  {
    bool ICanHasCheeseburger() const = 0;
  };
  using CheeseburgerComponentRequestBus = AZ::EBus<CheeseburgerComponentRequests>;
```

- Use the name format `MyComponentNotificationBus` for events that are broadcast from `MyComponent`, as in the following example.

  ```cpp
class CheeseburgerComponentNotifications : public AZ::ComponentBus
```
Provide Default Implementations of Methods

Notification buses typically provide default implementations of the methods within the interface. Many other components can monitor your component's events, but not all of them are interested in every event that your component sends. If you provide default implementations for all your methods, other components that subscribe to your events can implement only those events that are relevant to them.

EBus Event Naming

Good EBus event names are verbose. Classes can monitor multiple buses, so descriptive event names makes it clear which bus the function corresponds to. This practice also prevents potential name collisions among event interfaces from different buses.

The following example is a clearly named `PhysicsComponentNotificationBus` event.

```cpp
virtual void OnPhysicsEnabled() = 0;
```

The following example is an ambiguously named `PhysicsComponentNotificationBus` event.

```cpp
virtual void OnEnabled() = 0;
```

Avoid Using Type Definitions for Serialized Data

An instructive example from Lumberyard shows the importance of using classes instead of type definitions for serialized data. Formerly, `EntityId` used the type definition `uint32_t`. When the decision was made to change this to 64-bit, upgrade functions had to be written for every class that contained an `EntityId`. If `EntityId` had been a class, a single upgrade function could have been written for the class, and no further work would have been required. Obviously, this principle does not apply to primitive types like `bool`, `float`, `int`, and `string`. However, if you have a specific type that is serialized and might change in the future, implement it as a reflected class. This provides a single context where you can easily make the conversion for the class or type.

EBus Results

Always initialize a variable before calling an EBus event that overwrites the variable. Even if you are sure that a particular class or component is listening on the bus, it's worth handling the exceptional case. This is especially true in distributed environments in which entities can come and go as part of area-of-interest or other dynamic patterns.

The following example initializes a result variable before calling an EBus event that produces a result.

```cpp
AZ::Transform targetEntityTransform = AZ::Transform::Identity(); // initialize result variable...
EBUS_EVENT_ID_RESULT(targetEntityTransform, targetEntityId, AZ::TransformBus, GetWorldTM); // ...in case of no response
```

EBus Timing

Following are some best practices for the timing of EBus actions.
• In the Activate() (p. 976) function, make sure that connecting to buses is the last step.
• In the Deactivate() (p. 976) function, make sure that disconnecting from buses is the first step.
• In a multithreaded environment, it's possible to receive bus events from the moment that you connect to the bus until the moment you disconnect. For this reason, make sure of the following:
  • Your component is fully activated before it starts reacting to events.
  • Your component stops receiving events before it starts deactivation.

This practice prevents your component from being in a half-activated state when it starts reacting to events, or in a half-deactivated while still receiving events.

• When you send events on a notification-style bus, the last step in a function should ensure that the data is fully populated.

The following is an example to avoid.

```cpp
EBUS_EVENT_ID(GetEntityId(), OnTransformChanged, newTransform);
m_transform = newTransform;
```

If a component is monitoring the OnTransformChanged event and sets your transform in response to the event, the component's action will be undone by the `m_transform = newTransform;` assignment.

### Making Functions Public or Protected

Consider the following when deciding to make functions public or private.

• Make your bus functions **public** if they constitute the public interface for your class. While it's discouraged, Lumberyard does not prevent users from getting direct pointers to components and calling functions directly. To avoid this, make sure that your useful functions are public. For example, `MyComponent` should probably implement functions from `MyComponentRequestBus` publicly.

• Make your bus functions **protected** if they contain the private workings of your class. For example, your component's reaction to the `TransformNotificationBus::OnTransformChanged` event would likely be a private implementation detail.

### Additional Resources

For more information on components and EBuses, consult the following resources.

• For examples of EBus usage, see Usage and Examples (p. 1853).
• For in-depth information about EBuses, including conceptual diagrams, see Event Buses in Depth (p. 1857).
• For questions and answers regarding best practices for components and EBuses, see Components and EBuses: Questions and Answers (p. 1028).
• For C++ API reference documentation on the core EBus code, see the EBus API Reference in the Amazon Lumberyard C++ API Reference.

### Components and EBuses: Questions and Answers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The following are some questions and answers for using components and EBuses in Amazon Lumberyard.

**Q. Is it possible to mix Lumberyard’s legacy object entity system with its newer component entity system? If so, what is the best practice?**

**A.** If you enable legacy systems, the two systems can work side by side. However, no formal mechanisms exist to have them work together in the way most games would expect.

**Q. Are there combinations of components that should be avoided?**

**A.** In principle, no. However, every component has a small resource cost associated with it. When possible, avoid creating long chains of components if the desired objective be accomplished through simpler means.

**Q. When should I make an editor component?**

**A.** When editor–only functionality is required, or when the data in the editor must be different than the data in the runtime. For more information, see Editor Components (p. 984).

**Q. What is the best practice for a component to communicate with another component on the same entity? What if the component is on a different entity?**

**A.** Using EBuses for both events and event monitoring. For more information, see Working with the Event Bus (EBus) system (p. 1851).

**Q. What is the best way to find component entities at runtime?**

**A.** The method depends on the use case. Here are some possible approaches:

- Follow a convention in which you use the Tag (p. 875) component to apply tags to specific types of entities. Then use the tags to find the entities that you want. Note that searching through all the entities in a large world can be resource intensive.
- Create a custom component that listens on a new EBus for an event. At an appropriate time, emit that event so that all entities can react to it appropriately.
- If a hard connection is possible, use a component that contains an entity reference. Then look up the entity directly by ID at runtime.

**Q. What is the best way to group component entities?**

**A.** Because the Entity Outliner (p. 462) in Lumberyard Editor shows entities in parent–child hierarchy, group entities under a parent whenever possible. This provides contextual organization.

**Q. What are best practices for organizing and working with a large number of component entities in Lumberyard Editor?**

**A.** Where appropriate, use slices (p. 510) for repeated content. To make it easy to collapse or hide large groups of entities in the Entity Outliner, create a hierarchy of entities.

**Q. How should I decide whether to place code in a component constructor, in an OnInit, or in an OnActivate method?**

**A.** Use the following criteria:

- **Constructor:** Use for base initialization. A constructor is called only once, when the class is created.
- **OnInit:** Called only once, after all components have been created. Like a constructor, an OnInit method should contain only initialization code that is only required once. However, OnInit methods might be more appropriate in some cases for late binding or late initialization.
- **OnActivate:** Use this function for any task that needs to be set up or initialized every time the entity is activated. Entity activation occur multiple times depending on editor or game logic and the actions taken. Commonly, this includes tasks like connecting to EBuses so that the entity can be notified when events occur. Corresponding disconnects occur in DeActivate.
Q. What are best practices for adding or removing components at runtime?

A. Adding or removing components at runtime requires that the entity to which the component belongs be deactivated and then reactivated. This process is not resource intensive, but you must ensure that state is properly maintained by saving it before deactivation and restoring it after activation.

Q. What are best practices for creating entities and adding components at runtime using C++?

A. Only some Lumberyard components support runtime creation. Custom components must be specifically built to support runtime creation. For a discussion of the differences between editor and runtime components, see Editor Components (p. 984).

Gameplay Bus

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The gameplay bus alerts gameplay systems that an event has occurred. You can use the bus to send contextual messages between the visual scripting, scripting, and code parts of your game in a generic and extensible way.

The gameplay bus passes an `AZStd::any`, a class that uses type erasure to hold any C++ reflected type or any Lua primitive except for tables (string, number, Boolean, etc.). It includes mechanisms for type safety to ensure that it returns the same type that it is passed. In Lua, the type can be inspected if it has been exposed to the behavior context through the `typeid()` function, which uses the `AZ_RTTI` system.

Example

```lua
function MyComponent:OnEventBegin(param)
    if typeid(param) == typeid(Uuid) then
        -- param is a Uuid
    elseif typeid(param) == typeid(Vector3) then
        -- param is a Vector3
    end
end
```

**GameplayNotificationId**

The `GameplayNotificationId` is the type used as the ID for the gameplay bus. Its syntax is as follows.

```lua
GameplayNotificationId(const AZ::EntityId& entityChannel, AZ::Crc32 actionNameCrc, const AZ::Uuid& payloadType)
```

This function takes the `entityChannel`, `actionNameCrc`, and `payloadType` parameters and creates a unique ID that you can use to communicate with the bus.
entityChannel

When you write your events, choose the ID of an entity channel that makes sense in the context of your game. Components are automatically aware of the following two entity IDs, which you can use for channels. These IDs do not require an entity reference.

1. The component's own entity ID. To obtain it, call GetEntityId().
2. The default AZ::EntityId().

To communicate directly to a specific entity, use GetEntityId(). To communicate indirectly to a generic audience, use AZ::EntityId().

actionNameCrc

You can pass the actionNameCrc parameter as a string or as the AZ::Crc32 of that string when you construct the ID. This parameter should be the name of the event that gives context to the event data.

payloadType

Use the payloadType parameter to specify the type that is sent or received. This parameter is required starting in Lumberyard version 1.12.

GameplayNotifications

The GameplayNotifications class contains the gameplay bus type traits. It establishes the GameplayNotificationId as the bus ID. It defines the following events.

```c
void OnEventBegin(const AZStd::any& value)
void OnEventUpdate(const AZStd::any& value)
void OnEventEnd(const AZStd::any& value)
```

Script Examples

The Lua script examples in this section illustrate the use of GameplayNotificationBus to control what happens when an entity enters and then exits lava.

1. The following example implements the OnActivate function when the entity enters lava.

```lua
local InLavaBehavior = {
  Properties = {
  },
}

function InLavaBehavior:OnActivate()
  local gameplayId = GameplayNotificationId(self.entityId, "InLava")
  self.gameplayBus = GameplayNotificationBus.CreateHandler(self, inputBusId)
end
```

2. The following example implements the OnEventBegin function to start an animation associated with the event.

```lua
local InLavaBehavior = {
  Properties = {
  },
}

function InLavaBehavior:OnEventBegin()
  local gameplayId = GameplayNotificationId(self.entityId, "InLava")
  self.gameplayBus = GameplayNotificationBus.CreateHandler(self, inputBusId)
end
```
function InLavaBehavior:OnEventBegin(floatValue)
    local animInfo = AnimatedLayer("LavaHotFootDance", 0, true, 1.0, 0.0);
    SimpleAnimationComponentRequestBus.Event.StartAnimation(self.entityId, animInfo)

    -- tell the HUD to transition the screen effect to "In Lava"
    local gameplayId = GameplayNotificationId(EntityId(), "TransitionScreenEffect")
    GameplayNotificationBus.Event.OnEventBegin(gameplayId, "In Lava")
end

3. The following example implements the OnEventUpdating function to update a health component regarding the status of the entity.

function InLavaBehavior:OnEventUpdating(floatValue)
    -- alert the health component (this gameplay component is an example only) that we are taking damage, it can handle any death transitions
    HealthComponentBus.Event.TakeDamage(self.entityId, floatValue)
end

4. The following example implements the OnEventEnd function to end the animation and return status to normal.

function InLavaBehavior:OnEventEnd(floatValue)
    local animInfo = AnimatedLayer("Idle", 0, true, 1.0, 0.0);
    SimpleAnimationComponentRequestBus.Event.StartAnimation(self.entityId, animInfo)

    -- tell the HUD to transition the screen effect to "Normal"
    local gameplayId = GameplayNotificationId(EntityId(), "TransitionScreenEffect")
    GameplayNotificationBus.Event.OnEventBegin(gameplayId, "Normal")
end

5. The following example implements the OnDeactivate function to disconnect from the gameplay bus.

function InLavaBehavior:OnDeactivate()
    self.gameplayBus:Disconnect()
end

return InLavaBehavior

Reflecting Lumberyard Classes, Methods, and EBus Interfaces

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Lumberyard's reflection system to expose runtime code for C++-based objects, for Lumberyard Editor, and for scripting.

Lumberyard provides the following reflection contexts for these purposes:

- **Serialization Context (p. 1033)** – Provides persistence for C++ based objects through reflection for serialization.
Serialization Context

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the serialization context (`\dev\Code\Framework\AzCore\AzCore\Serialization\SerializeContext.*`) to provide persistence for C++ objects or any Lumberyard type. To implement this, make an `AzTypeInfo` declaration or use `AZ_RTTI` (runtime type information), as in the following example:

```cpp
class SerializedObject
{
public:
    AZ_RTTI(SerializedObject, "");
    static void Reflect(AZ::ReflectContext* context)
    {
        SerializeContext* serializeContext = azrtti_cast<SerializeContext*>(reflection);
        if (serializeContext)
        {
            serializeContext->Class<SerializedObject>()
            ;
        }
    }
};
```

You can also reflect native types and POD structs for serialization by creating an `AZ_TYPE_INFO` specialization, as in the following code example:

```cpp
AZ_TYPE_INFO_SPECIALIZE(AZStd::chrono::system_clock::time_point, "{5C48FD59-7267-405D-9C06-1EA31379FE82}");
AZ_TYPE_INFO_SPECIALIZE(float, "{EA2C3E90-AFBE-44d4-A90D-FAAF79BAF93D}");
```

Fields

To associate a text string with the address to a field of a serialized object, use the `Field` function, as in the following example. You can use the builder pattern to serialize multiple fields.

```cpp
serializedContext->Class<SerializedObject>()
    ->Field("myIntField", &SerializedObject::myIntField)
    ->Field("myFloatField", &SerializedObject::myFloatField)
    ;
```

Serializers

Serializers are a useful way to provide custom data formats. If you want to do custom processing on an object before writing or reading it, you can override Lumberyard's default serializer.
To override the default serializer, implement the `AZ::SerializeContext::IDataSerializer` interface. Use the interface to override how data is handled as it is streamed into its persistent format. You can also use the interface to determine the actions that occur when the reflected object is serialized (read or written).

The `AZ::Uuid` class (\dev\Code\Framework\AzCore\AzCore\Math\MathReflection.cpp) provides a good example of a custom serializer. To save a UUID value, the code writes it directly into the stream. This part of the code is straightforward.

```
/// Store the class data into a binary buffer
size_t Uuid::Save(const void* classPtr, IO::GenericStream& stream, bool)
{
    const Uuid* uuidPtr = reinterpret_cast<const Uuid*>(classPtr);
    return static_cast<size_t>(stream.Write(16, reinterpret_cast<const void*>(uuidPtr->data)));
}
```

Loading a UUID is also straightforward, but the code does some error checking to ensure that the data is loaded as expected:

```
/// Load the class data from a stream.
bool Uuid::Load(void* classPtr, IO::GenericStream& stream, unsigned int /*version*/, bool)
{
    if (stream.GetLength() < 16)
    {
        return false;
    }
    Uuid* uuidPtr = reinterpret_cast<Uuid*>(classPtr);
    if (stream.Read(16, reinterpret_cast<void*>(&uuidPtr->data)) == 16)
    {
        return true;
    }
    return false;
}
```

The custom serializer has functions that convert the data between binary and text formats. By converting the data into text format, you can store it in `.xml` or `.json` files.

The following `DataToText` function reads the binary value for the UUID from the incoming stream. The function converts the binary value into an `AZStd::string` and then writes it to the outgoing stream.

```
size_t Uuid::DataToText(IO::GenericStream& in, IO::GenericStream& out, bool)
{
    if (in.GetLength() < 16)
    {
        return 0;
    }
    Uuid value;
    void* dataPtr = reinterpret_cast<void*>(&value.data);
in.Read(16, dataPtr);
    char str[128];
    value.ToString(str, 128);
    AZStd::string outText = str;
    return static_cast<size_t>(out.Write(outText.size(), outText.data()));
}
```

The following `TextToData` function converts the text input string into binary UUID format and then writes the binary data out to the stream.

```
/// Convert text data to binary to support the loading of legacy formats.
```
Serialization Context

// Respect the text version if the text->binary format has changed!
size_t Uuid::TextToData(const char* text, unsigned int, IO::GenericStream& stream, bool)
{
    Uuid uuid = Uuid::CreateString(text);
    stream.Seek(0, IO::GenericStream::ST_SEEK_BEGIN);
    return static_cast<size_t>(stream.Write(16, uuid.data));
}

Data Containers

To create custom serialization for templates and types that are not directly reflected through the 
SerializeContext::Reflect function, you can use data containers.

To create a data container, implement the AZ::SerializeContext::IDataContainer interface. You 
can use this interface to provide serialization for a class or template of classes and let the user choose 
the elements to be serialized. This is possible because IDataContainer allows the user to override an 
EnumElements function. The EnumElements function determines which elements of the serialized class 
are enumerated and are therefore capable of being serialized.

Templates

Data containers provide the best way to add support for templates to the serialization context. The 
following templates have a metaclass that implements the IDataContainer interface and serializes the 
templates.

AZStd::vector<T>
AZStd::basic_string<T>
AZStd::unique_ptr<T>

Nontemplate Types

You can use the IDataContainer interface to serialize nontemplate types like AZStd::any. This is 
because the type of element that is serialized is dependent on the type that is stored in the AZStd::any 
object.

Stable Elements

Elements are considered stable if their pointers do not change when other elements are added to or 
removed from a container. Lumberyard's implementation of stable elements corresponds to the C++ 
+17 rules for iterator invalidation as documented in section 26 of the ISO/IEC 14882:2017(E) standard. 
The elements in types like AZStd::vector are not stable because they are stored in a contiguous 
sequence. When an element that is not at the end of the vector is removed, all elements after it in 
memory must shift to the left to keep the sequence contiguous. Stable elements can be removed from 
a container without affecting other elements in the container. You can use the IsStableElements 
function to determine the status of a container's elements. If a container's elements are not stable, you 
must enumerate them in order for them to be serialized.

The following code example shows how to set up serialization for a container that stores a dynamic 
sequence of homogenous elements.

template<class T, bool IsStableIterators>
class AZStdBasicContainer
    : public SerializeContext::IDataContainer
{
    public:
        typedef typename T::value_type ValueType;
        typedef typename AZStd::remove_pointer<typename T::value_type>::type ValueClass;
        //... Functions implementing the IDataContainer interface
A `SerializeContext::ClassElement` is a struct that uniquely identifies a serialized element of a class. It includes fields like the following:

- **TypeId** – an ID for looking up data in `ClassData` within the `SerializeContext`.
- **Name, NameCrc** – The name and CRC with which the element is serialized.
- **Element-specific serialization attributes**

To look up the name of the `SerializeContext::ClassElement` that the data container supports, override the `GetElement` function, as in the following example.

```cpp
// Returns the class element by looking up the CRC value of the element.
// Returns null if the element with the specified name can't be found.
const SerializeContext::ClassElement* GetElement(u32 elementNameCrc) const override
{
    if (elementNameCrc == m_classElement.m_nameCrc)
    {
        return &m_classElement;
    }
    return nullptr;
}

// The following GetElement method uses the supplied DataElement object to lookup the
// ClassElement with the supplied parameter. Returns true if it finds a ClassElement.
bool GetElement(SerializeContext::ClassElement& classElement, const
SerializeContext::DataElement& dataElement) const override
{
    if (dataElement.m_nameCrc == m_classElement.m_nameCrc)
    {
        classElement = m_classElement;
        return true;
    }
    return false;
}
```

The following example shows how to override the `EnumElement` method to specify the elements that are enumerated. Enumerating them enables them to be saved.

```cpp
/// Enumerate elements in the array.
/// The ElementCB callback enumerates the children of the elements in the array.
/// By invoking the callback on an element, the enumeration continues down the path for
/// that element.
void EnumElements(void* instance, const ElementCB& cb) override
{
    T* arrayPtr = reinterpret_cast<T*>(instance);
    typename T::iterator it = arrayPtr->begin();
    typename T::iterator end = arrayPtr->end();
    for (; it != end; ++it)
    {
        ValueType* valuePtr = &*it;
        if (!cb(valuePtr, m_classElement.m_typeId, m_classElement.m_genericClassInfo ?
        m_classElement.m_genericClassInfo->GetClassData() : nullptr, &m_classElement))
        {
            break;
        }
    }
}
```

To make a template editable in Lumberyard Editor and the reflected property editor, override the constraint functions in the following code:
// The following code defines the characteristics of the container that is serialized. // The editing facilities use this information to determine how to edit the elements within the container.

/// Return the number of elements in the container.
size_t  Size(void* instance) const override
{
    const T* arrayPtr = reinterpret_cast<const T*>(instance);
    return arrayPtr->size();
}
/// Return the capacity of the container. Return 0 for objects without fixed capacity.
size_t  Capacity(void* instance) const override
{
    (void)instance;
    return 0;
}
/// Return true if the element pointers do not change when the element is added to or removed from the container. If false, you MUST enumerate all elements.
bool  IsStableElements() const override
    { return IsStableIterators; }
/// Return true if the container has a fixed size; otherwise false.
bool  IsFixedSize() const override
    { return false; }
/// Return true if the container has a fixed capacity; otherwise false.
bool  IsFixedCapacity() const override
    { return false; }
/// Return true if the container is a smart pointer.
bool  IsSmartPointer() const override
    { return false; }
/// Return true if the elements can be retrieved by index.
bool  CanAccessElementsByIndex() const override
    { return false; }

Notes

• When IsFixedSize and IsFixedCapacity are false, the plus (+) and minus (–) buttons in the property editor can be used to add and remove elements from the data container.
• When IsSmartPointer is false, the data container does not create an instance of theSmartPointer type when an element is added to the container.
• When CanAccessElementsByIndex is false, the serialization system checks whether to allocate memory for new elements. CanAccessElementsByIndex is true for fixed-size containers like AZStd::array, AZStd::pair, and AZStd::tuple because those containers already have memory storage allocated for their elements.

To load an element into the template class instance, override the ReserveElement, StoreElement and RemoveElements functions, as in the following example.

/// Use the reserve element function.
/// The reserve element function allows creation of the element on the data container instance.
/// The following code serializes an element and returns an address to the reserved element.
void*  ReserveElement(void* instance, const SerializeContext::ClassElement* classElement) override
{
    (void)classElement;
    T* arrayPtr = reinterpret_cast<T*>(instance);
    arrayPtr->push_back();
    return &arrayPtr->back();
}
/// Use the GetElementByIndex function to get an element’s address by its index.
// Call this function before the element is loaded.
void* GetElementByIndex(void* instance, const SerializeContext::ClassElement* classElement, size_t index) override
{
    (void)instance;
    (void)classElement;
    (void)index;
    return nullptr;
}

/// Use the store element function.
void StoreElement(void* instance, void* element) override
{
    (void)instance;
    (void)element;
    // Do nothing; you have already pushed the element.
    // However, you can assert and check if the element belongs to the container.
}

/// Remove the element from the container.
/// This also deletes the memory associated with the element.
bool RemoveElement(void* instance, const void* element, SerializeContext* deletePointerDataContext) override
{
    T* arrayPtr = reinterpret_cast<T*>(instance);
    for (typename T::iterator it = arrayPtr->begin(); it != arrayPtr->end(); ++it)
    {
        void* arrayElement = &(*it);
        if (arrayElement == element)
        {
            if (deletePointerDataContext)
            {
                DeletePointerData(deletePointerDataContext, &m_classElement, arrayElement);
            }
            arrayPtr->erase(it);
            return true;
        }
    }
    return false;
}

/// Remove elements (remove an array of elements) whether the container is stable or not.
/// Stability can be tested by IsStableElement.
size_t RemoveElements(void* instance, const void** elements, size_t numElements, SerializeContext* deletePointerDataContext) override
{
    if (numElements == 0)
    {
        return 0;
    }
    size_t numRemoved = 0;
    // Handle the case when the container does not have stable elements.
    if (!IsStableElements)
    {
        // If the elements are in order, you can remove all of them from the container.
        // Otherwise, they must be sorted again locally (not done in this example).
        // Or, ask the user to pass the elements in order and remove the first N possible
        // in order.
        for (size_t i = 1; i < numElements; ++i)
        {
            if (elements[i - 1] >= elements[i])
            {
                AZ_TracePrintf("Serialization", "RemoveElements for AZStd::vector will
                perform optimally when the elements (addresses) are sorted in accending order!");
                numElements = i;
            }
        }
        // Traverse the vector in reverse order, and then addresses of elements that should
        not change.
    }
Using the DataContainer to Serialize a Template Class

After you have defined a data container, you can use it to serialize a specific type. For example, to set up serialization for the templated `AZStd::vector<T>`, you must serialize `SerializeGenericTypeInfo<T>` for `AZStd::vector`. To create the class data structure, you use the following `Create<ContainerType>` function:

```cpp
SerializeContext::ClassData::Create<ContainerType>("AZStd::vector", GetSpecializedTypeId(), Internal::NullFactory::GetInstance(), nullptr, &m_containerStorage);
```

The `Create<ContainerType>` function parameters are explained in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;AZStd::vector&quot;</td>
<td>Specifies the user friendly name of the class in a JSON or XML stream.</td>
</tr>
<tr>
<td>GetSpecializedTypeId</td>
<td>Creates an ID that enables serialization of different types. For example, an <code>AZStd::vector</code> of integers can be serialized as a type that is different from an <code>AZStd::vector</code> of floats. The unique ID is made by aggregating the template type <code>AZStd::vector</code> with the contained type <code>T</code>.</td>
</tr>
<tr>
<td>Internal::NullFactory</td>
<td>Used to prevent heap memory from being used to create an <code>AZStd::vector</code>. To load an <code>AZStd::vector</code> element of a pointer type, change this to <code>Serialize::InstanceFactory&lt;AZStd::vector&lt;T,A&gt;&gt;</code>.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nullptr</td>
<td>This is the Serializer parameter. Because the serialization occurs through a data container, this parameter is nullptr.</td>
</tr>
<tr>
<td>&amp;m_containerStorage</td>
<td>The m_containerStorage structure is an AZStdBasicContainer that ClassData uses to serialize the AZStd::vector element array.</td>
</tr>
</tbody>
</table>

The following code example uses the Create<ContainerType> function to set up serialization for the templated AZStd::vector<T>.

```cpp
/// Generic serialization example for AZStd::vector.
template<class T, class A>
struct SerializeGenericTypeInfo< AZStd::vector<T, A> >
{
    typedef typename AZStd::vector<T, A> ContainerType;
    class GenericClassInfoVector
    :
        public GenericClassInfo
    {
        public:
            AZ_TYPE_INFO(GenericClassInfoVector, "{2BADE35A-6F1B-4698-B2BC-3373D010020C}");
            GenericClassInfoVector()
        { // The following code creates the ClassData structure that specifies how an element is serialized.
            m_classData =
                SerializeContext::ClassData::CreateContainerType("AZStd::vector", GetSpecializedTypeId(), Internal::NullFactory::GetInstance(), nullptr, &m_containerStorage);
            SerializeContext::ClassData* GetClassData() override
            { return &m_classData; }
            size_t GetNumTemplatedArguments() override
            { return 1; }
            const Uuid& GetTemplatedTypeId(size_t element) override
            { (void)element;
                return SerializeGenericTypeInfo<T>::GetClassTypeId(); }
            const Uuid& GetSpecializedTypeId() const override
            { return azrtti_typeid<ContainerType>(); }
            const Uuid& GetGenericTypeId() const override
            { return TYPEINFO_Uuid(); }
            void Reflect(SerializeContext* serializeContext)
            { if (serializeContext)
                { serializeContext->RegisterGenericClassInfo(GetSpecializedTypeId(), this, &AnyTypeInfoConcept<ContainerType>::CreateAny);
                    if (GenericClassInfo* containerGenericClassInfo =
                        m_containerStorage.m_classElement.m_genericClassInfo)
                    { containerGenericClassInfo->Reflect(serializeContext); }
                }
            }
            static GenericClassInfoVector* Instance()   
```

Version 1.28
1040
Serialization Context

```cpp
static GenericClassInfoVector s_instance;
return &s_instance;
}
Internal::AZStdBasicContainer<ContainerType, false> m_containerStorage;
SerializeContext::ClassData m_classData;
};
static GenericClassInfo* GetGenericInfo()
{
return GenericClassInfoVector::Instance();
}
static const Uuid& GetClassTypeId()
{
return GenericClassInfoVector::Instance()->m_classData.m_typeId;
}
```

Events

To process data before or after you read or write serialized data, you can write serialization event handlers. For example, by handling serialization events, you can perform runtime initializations specific to the data that is serialized.

To create a serialization event handler, implement the `AZ::SerializeContext::IEventHandler` interface as in the following example.

The example uses an event handler to update a map container within the `SceneData` class after a `SceneData` instance has been serialized.

```cpp
class SceneDataEventHandler : public AZ::SerializeContext::IEventHandler
{
public:
    /// Rebuild the endpoint map.
    void OnWriteEnd(void* classPtr) override
    {
        auto* sceneData = reinterpret_cast<SceneData*>(classPtr);
        BuildEndpointMap(*sceneData);
    }
};
```

Data Overlays

You can use the serialization context to provide data from an external source during serialization. These external sources of data are called *data overlays*.

To create a data overlay, you implement an EBus (p. 1851) through which the data is serialized. The following example is the code that implements unit testing for the data overlay feature (\dev\Code\Framework\AzCore\Tests\Serialization.cpp):

```cpp
struct DataOverlayTestStruct
{
```
The DataOverlayTestStruct holds data fields to be reflected for serialization:

```
serializeContext.Class<DataOverlayTestStruct>()
  ->Field("int", &DataOverlayTestStruct::m_int)
  ->Field("intVector", &DataOverlayTestStruct::m_intVector)
  ->Field("pointer", &DataOverlayTestStruct::m_ptr);
```

Next, implement the data overlay provider. The provider represents the data source that is overlaid into the serialized data.

The following code shows an example of a data overlay provider:

```
class DataOverlayProviderExample
 : public DataOverlayProviderBus::Handler
{
public:
  static DataOverlayProviderId GetProviderId() { return AZ_CRC("DataOverlayProviderExample", 0x60dafdbd); }  
  static u32 GetIntToken() { return AZ_CRC("int_data", 0xd74868f3); }  
  static u32 GetVectorToken() { return AZ_CRC("vector_data", 0x0aca20c0); }  
  static u32 GetPointerToken() { return AZ_CRC("pointer_data", 0xa46a746e); }
  DataOverlayProviderExample()
  {
    m_ptrData.m_int = 5;
    m_ptrData.m_intVector.push_back(1);
    m_ptrData.m_ptr = nullptr;
    m_data.m_int = 3;
    m_data.m_intVector.push_back(10);
    m_data.m_intVector.push_back(20);
    m_data.m_intVector.push_back(30);
    m_data.m_ptr = &m_ptrData;
  }
  void FillOverlayData(DataOverlayTarget* dest, const DataOverlayToken& dataToken) override
  {
    if (*reinterpret_cast<const u32*>(dataToken.m_dataUri.data()) == GetIntToken())
      { dest->SetData(m_data.m_int); }
    else if (*reinterpret_cast<const u32*>(dataToken.m_dataUri.data()) == GetVectorToken())
      { dest->SetData(m_data.m_intVector); }
    else if (*reinterpret_cast<const u32*>(dataToken.m_dataUri.data()) == GetPointerToken())
      { dest->SetData(*m_data.m_ptr); }
    }
  DataOverlayTestStruct m_data;
  DataOverlayTestStruct m_ptrData;
};
```
**Serialization Context**

DataOverlayProviderExample uses the Crc32 ID for the reflected DataOverlayTestStruct source data fields. Then the example implements the DataOverlayProviderBus::Handler FillOverlayData function. The FillOverlayData function is where the actual data overlay occurs. The DataOverlayToken holds the ID of the field that is serialized. If the ID matches one of the fields that you want to overlay, you can use DataOverlayTarget to set the data.

### Versioning your Component Serialization

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

As requirements, code, and data representation change, you may need to modify your implementation of data reflection. However, changes to serialized data can result in incompatibilities. To manage compatibility, you can assign a version number increment to your serialized data structures. With this approach, you can perform validation during serialization to ensure that the data being read matches the format that the reflection system specifies. We recommend that you increase the version number of serialized data anytime there is a change to the reflected fields.

The following code shows how to specify a version number for the serialization context.

```cpp
serializeContext->Class<SerializedStruct>()
    ->Version(1)
```

Successful conversion of serialized data to a newer version requires careful planning. Use the information in the following sections to help with your planning.

- Version Converters (p. 1043)
- Upgrade Class Builders (p. 1045)
- Deprecation (p. 1049)

**Important**

If you upgraded to Lumberyard v1.23 or later from a previous version of Lumberyard, and you have not converted your slice files to the new slice file format, you must upgrade these files using the Slice Upgrade Pipeline before you can use the NameChange and TypeChange class builders. For more information about this upgrade tool and how to use it, see Converting Slices with the Slice Upgrade Pipeline (p. 529).

**Important**

If you are using Lumberyard v1.22 or earlier, be sure to read important information about Avoiding Data Loss when Serializing Component Data (p. 1049).

### Version Converters

A version change can create incompatibilities that require data to be converted from one format to another. To resolve this, you can implement a version converter that reformats data "on the spot" to maintain data compatibility. For example, you might require a version converter if you change a data type or a container (for example, an AZStd::vector becomes an AZStd::unordered_map).

Use the `Version` function mentioned in the previous section to specify a version converter, as in the following example.

```cpp
serializeContext->Class<EditorEntitySortComponent, EditorComponentBase>()
    ->Version(2, &SerializationConverter)
```
Version converters operate directly on the serialized data.

To facilitate the creation of version converters, Lumberyard provides helper functions and examples such as the following:

- To locate a specific element to manipulate, you can use the `AZ::Utils::FindDescendantElements` helper function.
- To access serialized data and manipulate it, you can use the public functions in the `DataElementNode` class (`\dev\Code\Framework\AzCore\AzCore\Serialization\SerializeContext.h`).
- For version converter examples, see the AZ core serialization unit test in the `\dev\Code\Framework\AzCore\Tests\Serialization.cpp` file.

A version conversion operation that replaces a container might follow this common pattern:

1. Compare the version number of the data being serialized with the current version. If the versions do not match, perform the steps that follow.
2. Locate the element to convert by its `Crc32` key.
3. Create a container to store the updated elements.
4. Populate the new container with the existing data.
5. Delete the old element from the root data.
6. Use the same `Crc32` key to add the new container as a new element in the root data.

After this operation is completed, the data exists in the new format. When the data is serialized again, it is stored in the latest format.

The following code shows an example of data conversion:

```cpp
if (rootElement.GetVersion() <= 1)
{
    // This line of code:
    //  using Events = AZStd::vector<EBusEventEntry>;
    //  is changed to this:
    //  using EventMap = AZStd::unordered_map<AZ::Crc32, EBusEventEntry>;
    auto ebusEventEntryElements = AZ::Utils::FindDescendantElements(serializeContext, rootElement, AZStd::vector<AZ::Crc32> {AZ_CRC("m_events", 0x191405b4), AZ_CRC("element", 0x41405e39)});
    EBusEventHandler::EventMap eventMap;
    for (AZ::SerializeContext::DataElementNode* ebusEventEntryElement : ebusEventEntryElements)
    {
        EBusEventEntry eventEntry;
        if (!ebusEventEntryElement->GetDataHierarchy(serializeContext, eventEntry))
        {
            return false;
        }
        AZ::Crc32 key = AZ::Crc32(eventEntry.m_eventName.c_str());
        AZ_Assert(eventMap.find(key) == eventMap.end(), "Duplicated event found while converting EBusEventHandler from version 1 to 2.");
        eventMap[key] = eventEntry;
    }
    // Remove the previous Events element.
    rootElement.RemoveElementByName(AZ_CRC("m_events", 0x191405b4));
    // Replace it with the new EventMap element.
    if (rootElement.AddElementWithData(serializeContext, "m_eventMap", eventMap) == -1)
    {
        return false;
    }
    return true;
}
```
Note
If you need to emit a warning or error when a conversion fails (for example, for asset builds), use the `AZ_Warning` or `AZ_Error` macro. For related source code, see `lumberyard_version\dev\Code\Framework\AzCore\AzCore\Debug\Trace.h`.

Upgrade Class Builders

Slice data patches present a unique challenge to versioning your component data structures. Data patches cannot be upgraded by version converters because they do not contain all the information about a component class. Changing the serialization of a component without upgrading data patches that contain partial component data can lead to crashes, corrupted slice data, or invalid slice files that cannot be loaded or manipulated and must be rebuilt from scratch.

In most cases, the solution is to use NameChange and TypeChange class builders alongside your version converters. This causes the serializer to update the data patch and apply basic type changes and field name changes. These builders can be chained together to upgrade across multiple version changes, and can also be written to skip versions entirely.

Class Builder Syntax

Name-change class builders require an input and output version, followed by the input serialized name and a new output name.

```
NameChange(InputVersion, OutputVersion, "OldFieldName", "NewFieldName")
```

Type-change class builders require input and output data types as template arguments, followed by the relevant field name, the input and output version, and a conversion function.

```
TypeChange<InputType, OutputType>("FieldName", InputVersion, OutputVersion, Function<OutputType(InputType)>)
```

NameChange Class Builder Examples

In the following example, we use a NameChange class builder to change a serialized name of a field from "MyData", used in version 4 of the component serialization, to "Data" in version 5.

```cpp
serializeContext->Class<ExampleClass>()
  ->Version(5)
  ->Field("Data", &ExampleClass::m_data)
  ->NameChange(4, 5, "MyData", "Data");
```

You can also change the serialized name of a struct or class member when reflecting a class to the serialize context. In the following example, we use a NameChange class builder to change the name from "MyStructData" in version 4, to "StructData" in version 5.

```cpp
class ExampleClass
{
```
Serialization Context

DataStruct m_data;
};

serializeContext->Class<ExampleClass>()
 ->Version(5)
 ->Field("StructData", &ExampleClass::m_data)
 ->NameChange(4, 5, "MyStructData", "StructData");

TypeChange Class Builder Examples

In the following example, class member m_data has changed from an int in version 4 to a float in version 5. We add a TypeChange class builder to the serialization context so that any data patches containing the serialized field name "MyData" will be applied using the new data type.

// Serialization Context for Version 4:
class ExampleClass
{
    ...
    int m_data;
    reflect(...)
    {
        serializeContext->Class<ExampleClass>()
        ->Version(4)
        ->Field("MyData", &ExampleClass::m_data);
    }
};

// Serialization Context for Version 5:
class ExampleClass
{
    ...
    float m_data;
    reflect(...)
    {
        serializeContext->Class<ExampleClass>()
        ->Version(5)
        ->Field("MyData", &ExampleClass::m_data)
        ->TypeChange<int, float>("MyData", 4, 5, [](int in)->float { return (float)in; });
    }
};

You can also handle nesting value changes. In the following example, the field m_data has become nested inside of the new MyData struct in version 5. We use a TypeChange class builder to instruct the serializer to convert the simple int data type to the more complex MyData type.

// Serialization Context for Version 4:
class ExampleClass
{
    ...
    int m_data;
    reflect(...)
    {
        serializeContext->Class<ExampleClass>()
        ->Version(4)
        ->Field("MyData", &ExampleClass::m_data);
    }
};
Advanced Class Builder Examples

The following examples demonstrate more complex usage of class builders.

**Example : Multiple Upgrades in One Version**

Type changes take priority over name changes. You can apply both in the same version upgrade, but the type change is applied first. Therefore, always specify the old field name in the `TypeChange` when changing both a type and a name at the same time.

In the following example, a `TypeChange` changes the type from `float` to `int`. It is immediately followed by a `NameChange` that changes the serialized name from "FloatData" to "IntData".

```cpp
// Serialization Context for Version 4:
class ExampleClass
{
    ... ...
    float m_data;
    static void Reflect(...)
    {
        serializeContext->Class<ExampleClass>()
            ->Version(4)
            ->Field("FloatData", &ExampleClass::m_data);
    }
};

// Serialization Context for Version 5:
class ExampleClass
{
    ... ...
    int m_data;
    static void Reflect(...)
    {
        serializeContext->Class<ExampleClass>()
            ->Version(5)
            ->Field("IntData", &ExampleClass::m_data)
            ->TypeChange<float, int>("FloatData", 4, 5, [](float in)->int { return (int)in; })
            ->NameChange(4, 5, "FloatData", "IntData");
    }
};
```
Example: Version Skipping

A `TypeChange` can skip multiple versions. Skipping versions should be used only when intermediate type changes contain conversions that could lose data.

In the following example, using `ExampleClass`, the member variable `m_data` changes from a `float` in version 1 to an `int` in version 2. Then in version 3, `m_data` changes back to a `float`. Multiple `TypeChange` class builders are used to avoid losing the floating point precision when upgrading older overrides to version 3, while still providing the ability to fix data patches written using version 2 of `ExampleClass`.

```cpp
// Version 1 of ExampleClass:
class ExampleClass
{
    ...  
    float m_data;
    static void Reflect(...)
    {
        serializeContext->Class<ExampleClass>()
            ->Version(1)
            ->Field("Data", &ExampleClass::m_data);
    }
};

// Version 2 of ExampleClass:
class ExampleClass
{
    ...
    int m_data;
    static void Reflect(...)
    {
        serializeContext->Class<ExampleClass>()
            ->Version(2)
            ->Field("Data", &ExampleClass::m_data)
            ->TypeChange<float, int>("Data", 1, 2, [](float in)->int { return (int)in; });
    };

// Version 3 of ExampleClass:
class ExampleClass
{
    ...
    float m_data;
    static void Reflect(...)
    {
        serializeContext->Class<ExampleClass>()
            ->Version(3)
            ->Field("Data", &ExampleClass::m_data)
            ->TypeChange<float, int>("Data", 1, 2, [](float in)->int { return (int)in; })
            ->TypeChange<int, float>("Data", 2, 3, [](int in)->float { return (float)in; })
            ->TypeChange<float, float>("Data", 1, 3, [](float in)->float { return in; });
    }
};
```

**Note**

We strongly recommend that you do not use this version skipping technique with the `NameChange` builder. Doing so will cause problems for any `TypeChange` builders used on the same field in between the skipped versions as they try to match the serialized field name.
Deprecation

The serialization context also supports deprecation of a previously reflected class name. To deprecate a class, use the `ClassDeprecate` method. After a class is deprecated, any instances of the class are silently discarded during load.

The following example shows the use of the `ClassDeprecate` method.

```c++
serializeContext->ClassDeprecate("DeprecatedClass", "{893CA46E-6D1A-4D27-94F7-09E26DE5AE4B}");
```

Avoiding Data Loss when Serializing Component Data

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

There are ways that you can avoid data loss when serializing component data. This topic describes how to deal with slice data patch issues when using version converters, and provides some best practices for avoiding data loss.

Avoiding Data Loss and Maintaining Stability in Lumberyard v1.22 and Earlier

This section provides specific information and best practices for avoiding data loss in versions of Lumberyard up through and including v1.22. First, it describes what you need to keep in mind when using version converters, and how to help avoid subsequent data loss in your slice data patches. Then, it provides a two-step process for avoiding data loss and maintaining stability.

**Note**
In Lumberyard v1.23, the issues described in the following topics were addressed with the introduction of the new slice file format and the NameChange and TypeChange class builders.
For more information about these class builders and how to upgrade your slice file format, see Versioning your Component Serialization (p. 1043).

**Topics**
- The Data Patching Issue  (p. 1049)
- How to Avoid Data Loss  (p. 1051)

The Data Patching Issue

During development, components undergo significant changes. Member variables are added, removed, renamed, and changed. Consequentially, serialization of the component also changes. When any change affects serialization, a version converter function can modify older data so that it can continue working with the new object state.

**Note**
Version converter functions work only when an entire class is supplied.

However, there are some scenarios where an entire class is not available. For example, consider data patches in slice files, which represent the difference between two serializable objects. These patches can be as small as a single serialized value or include entire classes. In the cases where they don't have a full class worth of information, the information in a data patch cannot be run through a version converter function. This means that when serialization of a component changes, any data patches stored in nested slices that apply to those components are no longer valid and cannot be recovered.
The following is an example of a version change that could lead to data loss. This example starts with a basic component that has just enough information to be reflected. It uses an integer with a value between 1 and 100 to represent the current blend state.

```cpp
// Blend Component (V1)
class BlendComponent : public Component
{
public:

AZ_TYPE_INFO(BlendComponent, "{ED986571-2B3D-4E22-A2D8-F74A29A730DE}");
static void Reflect(ReflectContext* context)
{
    if (auto serializeContext = azrtti_cast<SerializeContext*>(context))
    {
        serializeContext->Class<BlendComponent>()
            ->Version(1)
            ->Field("BlendState", &BlendComponent::m_blendState);
    }
}

private:

// A value from 1 to 100 representing the blend state
int m_blendState;
};
```

At some point during development, it's decided that additional resolution is required for the blend component. So the integer is changed to a floating point value with a valid range between 0.0f and 1.0f to represent the current blend state. There are a lot of assets that were saved with the original integer value, so a version converter is added which performs this upgrade automatically.

```cpp
// Blend Component (V2)
class BlendComponent : public Component
{
public:

AZ_TYPE_INFO(BlendComponent, "{ED986571-2B3D-4E22-A2D8-F74A29A730DE}");
static void Reflect(ReflectContext* context)
{
    if (auto serializeContext = azrtti_cast<SerializeContext*>(context))
    {
        serializeContext->Class<BlendComponent>()
            ->Version(2, BlendComponent::VersionUpgrader)
            // /// SERIALIZATION OF THE FIELD HASN'T CHANGED ////
            ->Field("BlendState", &BlendComponent::m_blendState);
    }
}

///// ONLY WORKS WITH A CLASS ELEMENT REPRESENTING THE ENTIRE BLEND COMPONENT TYPE /////

bool VersionUpgrader(SerializeContext& context, DataElementNode& classElement)
{
    if(version < 2)
    {
        int oldBlendValue;
        int oldBlendIndex = classElement.FindElement(AZ_CRC("BlendState"));
        if(oldBlendIndex != -1)
        {
            if(classElement.GetSubElement(oldBlendIndex).GetData(oldBlendValue))
            {
                float newBlendValue = float(oldBlendValue) / 100.0f;
                classElement.RemoveElement(oldBlendIndex);
                classElement.AddElementWithData(context, "BlendState", newBlendValue);
            }
        }
    }
    return true;
}
```
This automatic conversion works great when loading all slices and levels containing entities with old BlendComponents in them. The earlier versions are automatically modified to properly take advantage of the new floating point format. However, this conversion function is not effective when data patches are involved. Since data patches do not contain a representation of the entire BlendComponent class, the version converter cannot be used to modify the lone value stored in the data patch. In fact, there is no way for the serializer to know that the integral value being supplied is out of date. Behavior at this point is undefined and could result in a crash or, in some cases, data loss and data corruption.

The following example shows the sequence of events that can cause the data loss or corruption.

**An example sequence of events**

1. BlendComponent Version 1 is created.
2. A content creator makes "Slice A," which includes an entity with BlendComponent Version 1 on it.
3. A content creator instantiates Slice A and changes the Blend Value field in the Blend Component.
4. The creator now selects Slice A, and possibly other entities, and creates a new nested slice, "Slice B."
5. Slice B now contains a data patch, which points to the Blend Value field in Slice A and contains an integer.
6. The BlendComponent is updated to Version 2. Blend Value is now stored as a float.
7. A content creator instantiates Slice B. This results in undefined behavior.

**How to Avoid Data Loss**

We recommend the following three-step process to help avoid data loss and stability issues in Lumberyard v1.22 and earlier:

1. Do not change existing serialization fields. If a change is required, add a new field. This prevents earlier data patches from potentially causing crashes and lets you continue loading old assets. When implementing a version converter, do not remove the old field. Just read the old field and add a new one.
2. In your component's Init function, perform the conversion again on the old data. If the result differs from the data in the new field, it must have come from a data patch, and you can replace it.
3. Propagate all changes made to the value in the new field over to the old field. This prevents the Init function from overwriting any future changes made to the new field's value when loading.

**Note**

If it is not possible to keep the two values in sync in step 3, then the only way to prevent further issues after following steps 1 and 2 is to load and resave all slices which contain the versioned component, and then remove the Init function and the old field.

The following example demonstrates these steps for preventing data loss by revising the earlier example of version 2 of the BlendComponent.
// Blend Component (V2)
class BlendComponent : public Component
{
public:
    AZ_TYPE_INFO(BlendComponent, "{ED986571-2B3D-4E22-A2D8-F74A29A730DE}");
    static void Reflect(ReflectContext* context)
    {
        if (auto serializeContext = azrtti_cast<SerializeContext*>(context))
        {
            serializeContext->Class<BlendComponent>()
                ->Version(2, BlendComponent::VersionUpgrader)
                ->Field("BlendState", &BlendComponent::m_blendState)
                // REFLECT THE NEW FIELD AND THE OLD FIELD /////
                ->Field("NewBlendState", &BlendComponent::m_floatBlendState);
        }
    }

    void Init() override
    {
        // MIGRATE DATA PATCHES TO NEW DATA HERE /////
        float newBlendState = float(m_blendState) / 100.0f;
        if (newBlendState != m_floatBlendState)
        {
            m_floatBlendState = newBlendState;
        }
    }

    bool VersionUpgrader(SerializeContext & context, DataElementNode & classElement)
    {
        if (version < 2)
        {
            int oldBlendValue;
            int oldBlendIndex = classElement.FindElement(AZ_CRC("BlendState"));
            if (oldBlendIndex != -1)
            {
                if (classElement.GetSubElement(oldBlendIndex).GetData(oldBlendValue))
                {
                    float newBlendValue = float(oldBlendValue) / 100.0f;
                    // NO LONGER REMOVE THE OLD DATA ///</
                    classElement.AddElementWithData(context, "NewBlendState",
                        newBlendValue);
                }
            }
        }
    }

    private:
    // A value from 1 to 100 representing the blend state
    // ADD A NEW FIELD RATHER THAN REPLACING THE OLD ONE /////
    int m_blendState;
    float m_floatBlendState;
};

Edit Context

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

The Lumberyard edit context is a utility context that relies on the Serialization Context (p. 1033). You can use the edit context to expose parameters of serialized data for editing in Lumberyard Editor.
Lumberyard User Guide Lumberyard User Guide
Behavior Context

However, the edit context is an abstract container for edit data. As such, it is not directly tied to any
speciﬁc editor. Any editor can query the data in the edit context and implement its own visualization and
editing controls.
The following code shows an EditContext deﬁnition:
AZ::EditContext* editContext = serializeContext->GetEditContext();
if (editContext)
{
editContext->Class<MyEditStruct>("MyEditStruct", "My edit struct class used for ...")
;
}

An EditContext consists of ClassElements and DataElements.
• ClassElements – Specify attributes of the class that was reﬂected through the
SerializeContext::Class. You can use this to group common elements.
• DataElements – Specify the display, behavior, and visualization of the ﬁelds that were serialized
through SerializeContext::Field.

Attributes
You can use the EditContext to add arbitrary attributes to class and data elements.
Attributes are template based. As such, they can be of any type, including functions, as in the following
example.
editContext->Class<EditorLightComponent>(
"Light", "Attach lighting to an entity.")
->ClassElement(AZ::Edit::ClassElements::EditorData, "")
->Attribute(AZ::Edit::Attributes::AutoExpand, true)
->Attribute(AZ::Edit::Attributes::NameLabelOverride,
&EditorLightComponent::GetLightTypeText)
->DataElement(AZ::Edit::UIHandlers::Default, &EditorLightComponent::m_configuration,
"Settings", "Light configuration")
->Attribute(AZ::Edit::Attributes::Visibility,
AZ_CRC("PropertyVisibility_ShowChildrenOnly", 0xef428f20))
->ClassElement(AZ::Edit::ClassElements::Group, "Cubemap generation")
->Attribute(AZ::Edit::Attributes::Visibility, &EditorLightComponent::IsProbe)
->Attribute(AZ::Edit::Attributes::AutoExpand, true)

For convenience, Lumberyard stores a library of frequently used and implemented attributes in the ﬁle
dev\Code\Framework\AzCore\AzCore\Serialization\EditContextConstants.inl.

Behavior Context
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.
You can use the behavior context to reﬂect runtime code to make it accessible to scripting languages like
Lua or scripting environments like Lumberyard's Script Canvas. The behavior context provides bindings
that invoke runtime C++ code.
You can use the behavior context to bind the following C++ constructs for scripting:
Version 1.28
1053


• Classes (p. 1054)
• Methods (p. 1056)
• Properties (p. 1057)
• Constants (p. 1057)
• Enums (p. 1057)

In addition to the bindings for C++ constructs, the behavior context provides bindings for Lumberyard's EBus (p. 1851).

• EBus (p. 1058)
• Event Handlers (p. 1058)
• Events (p. 1059)

**Classes**

Classes that are bound to the behavior context become objects that can be instantiated in a script environment. To reflect a class, you must provide the type that is reflected as a template argument to the class function. The base classes should also be specified, as in the following example:

```cpp
if (BehaviorContext* behaviorContext = azrtti_cast<BehaviorContext*>(reflection))
{
    behaviorContext->Class<MyClass, TheBaseClass>();
}
```

**Attributes**

You can use the built-in attributes described in this section to decorate a class.

**Category**

Used by the editor to categorize the object in a list.

Type: string

You can use the forward slash (/) separator to nest categories, as in the following example:

```cpp
Attribute(AZ::Script::Attributes::Category, "Gameplay/Triggers")
```

**ExcludeFrom**

A flag that optionally hides this object from specific types of exposure. This flag is primarily used for internal objects that are not intended to be accessible by script.

Type: AZ::Script::Attributes::ExcludeFlags

Possible values: List, Documentation, Preview, All

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Hides the object from editor lists.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Makes the object unavailable for self-documentation.</td>
</tr>
<tr>
<td>Preview</td>
<td>Excludes this object from appearing in preview builds but can be enabled manually.</td>
</tr>
</tbody>
</table>
### Behavior Context

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Hides the object from all possible uses.</td>
</tr>
</tbody>
</table>

#### Storage

Specifies the owner of the memory storage for the reflected object.

**Type:** AZ::Script::Attributes::StorageType

**Possible values:** ScriptOwn, RuntimeOwn, Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptOwn</td>
<td>Specifies that the memory is owned and maintained by the script system (for example, Lua). This is the default value.</td>
</tr>
<tr>
<td>RuntimeOwn</td>
<td>Specifies that the memory is managed by a native code implementation in C++.</td>
</tr>
<tr>
<td>Value</td>
<td>Specifies that the object is stored by value in the script system's virtual machine (VM).</td>
</tr>
</tbody>
</table>

#### ConstructibleFromNil

Specifies whether the class is constructed by default when nil is provided.

**Type:** bool

**Possible values:** true, false

#### ClassNameOverride

Provides a custom name for script reflection that is different from the behavior context name.

**Type:** string

#### Ignore

Specifies whether the element is used script reflection.

**Type:** bool

**Possible values:** true, false

#### Deprecated

Marks a reflected class, method, EBus, or property as deprecated.

**Type:** bool

**Possible values:** true, false

#### ToolTip

Used by the editor to display additional information in a tool tip.

**Type:** string

### Class Reflection Examples

Classes reflected for scripting become objects that can be instanced in Lua, as in the following example:
local myObj = MyClass()

Classes can also specify custom constructors for scripting purposes. For example, the following constructor enables a custom constructor that specifies an integer:

behaviorContext->Class<MyClass>("MyClass")
    ->Constructor<int>()
;

This constructor enables the class to be instantiated in Lua in the following way:

-- Constructor in Lua
local myClass = MyClass(10)

Methods

You can reflect methods either as free functions or as part of classes, as in the following examples:

// This method is reflected as a free function:
behaviorContext->Method("AZTestAssert", &AZTestAssert);

// This method is reflected as a part of a class:
behaviorContext->Class<MyMath>("MyMath")
    ->Method("Cos", &cosf);

As in the previous example, class methods that are reflected for scripting are accessible through the reflected class:

-- Method from a class
local math = MyMath()
local result = math:Cos(3.14)

-- Free method
AZTestAssert(ScriptClass ~= nil)

Methods can specify parameters and default values for those parameters, as in the following example:

int globalMethod(int a)
{
    return a + 3;
}
behaviorContext->Method("globalMethod", &globalMethod, BehaviorMakeDefaultValues(555))

When you bind a method to the behavior context, you can provide an array of strings that contain the method's name and describe its arguments. This is useful when you want to provide friendly names or facilitate documentation.

The following example shows a method whose arguments contain descriptive strings.

// Given this method:
bool BoundsCheckMethodWithDefaultValue(float value, float epsilon, float minBounds, float maxBounds)
{
    (void)epsilon;
    return value >= minBounds && value < maxBounds;
// Bind the given method to the behavior context with friendly argument names.
Method("MemberWithDefaultValues", &BehaviorTestClass::BoundsCheckMethodWithDefaultValue,
  { {{"value", "Value which will be checked to be within the two bounds arguments"},
    {"delta", "The epsilon value"},
    BehaviorMakeDefaultValue(0.1f)},
    {{"minBound", "The minimum bounds value"},
    BehaviorMakeDefaultValue(0.0f)},
    {{"maxBound", "The maximum bounds value"},
    BehaviorMakeDefaultValue(1.0f)}}
  );

This approach is especially useful in Lumberyard's Script Canvas so that users can understand the meaning of the arguments that they are expected to provide.

**Properties**

You can reflect properties as parts of classes or as global properties. You can reflect properties as write only, read only or read/write. To do this, provide either a getter, a setter, or both for the property.

The following table shows helper macros that you can use to simplify the specification of a property's getter or setter or both.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Helper</th>
<th>Macro Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setter</td>
<td>BehaviorValueSetter-&gt;Property(&quot;boolean&quot;, nullptr, BehaviorValueSetter(&amp;TestBehaviorContextProperties::m_booleanProp))</td>
<td></td>
</tr>
<tr>
<td>Getter</td>
<td>BehaviorValueGetter-&gt;Property(&quot;boolean&quot;, BehaviorValueGetter(&amp;TestBehaviorContextProperties::m_booleanProp), nullptr)</td>
<td></td>
</tr>
<tr>
<td>Both</td>
<td>BehaviorValueProperty-&gt;Property(&quot;isStatic&quot;, BehaviorValueProperty(&amp;AZ::TransformConfig::m_isStatic))</td>
<td></td>
</tr>
</tbody>
</table>

**Constants**

You can reflect constants into the behavior context. Constants are by definition read-only. The following examples reflect two constants into the behavior context:

```cpp
behaviorContext->Constant("SystemEntityId", BehaviorConstant(SystemEntityId));
behaviorContext->Constant("PI", BehaviorConstant(3.14f));
```

** Enums**

Enums are reflected globally into the behavior context. For this reason, it is important to give them unique names.

The following example reflects the enum `GE_VALUE1` and two of its possible values globally into the behavior context.

```cpp
behaviorContext->Enum<(int)GE_VALUE1>("GE_VALUE1");
behaviorContext->Enum<(int)GlobalClassEnum::Value1>("Value1");
```
EBus

The ability to bind an EBus to the behavior context enables scripting to become driven and modular. The two main use cases for reflecting EBuses to the behavior context are event handlers and events.

Because EBuses are highly configurable, the features that become available depend on how the EBuses was created. For more information, see the Working with the Event Bus (EBus) system (p. 1851).

Event Handlers

By reflecting event handlers, you can enable events to be implemented directly in script.

To implement an event handler

1. Given the following EBus, reflect an event handler:

   ```cpp
class TestBusMessages
   : public AZ::EBusTraits
   {
   public:
    virtual void SetSum1(int) = 0;
    virtual int GetSum1(int) = 0;
   };
   using TestBus = AZ::EBus<TestBusMessages>;
   ```

2. Implement the event handler bindings, as in the following example:

   ```cpp
class TestBusHandler
   : public TestBus::Handler
   , public AZ::BehaviorEBusHandler
   
   { public:
    AZ_EBUS_BEHAVIOR_BINDER(TestBusHandler, "{CD26E702-6F40-4FF9-816D-4DCB652D97DF}",
    AZ::SystemAllocator,
    SetSum1,
    GetSum1);
   void SetSum1(int d1) override
   {
    Call(FN_SetSum1, d1);
   }
   int GetSum1(int d1) override
   {
    int result = 0;
    CallResult(result, FN_GetSum1, d1);
    return result;
   }
   };
```

This handler is what binds a C++ EBus interface to a script language such as Lua.

3. Tell the behavior context reflection that the event handler is available, as in the following example.

   ```cpp
   behaviorContext.EBus<TestBus>("TestBus")-
   Handler<TestBusHandler>()->
   Event("SetSum1", &TestBus::Events::SetSum1)->
   Event("GetSum1", &TestBus::Events::GetSum1)->
   ;
   ```

4. Optionally provide the implementation for the EBus handler, as in the following example.
Lumberyard User Guide

Behavior Context

MyBusHandlerMetaTable1 = {
    SetSum1 = function(self, _1)
        -- custom handler code can go here!
        TestAssert(_1 == 1)
        end,
    GetSum1 = function(self, _1)
        -- custom handler code can go here!
        return _1
        end,
}

handler = TestBus.Connect(MyBusHandlerMetaTable1)

Events

Events provide a mechanism to broadcast or send an event directly to any object that is connected to the
EBus through an ID.

By reflecting events to an EBus, you can broadcast or send the events in scripting, as the following
examples show.

behaviorContext.EBus<TestBus>("TestBus") ->
    Handler<TestBusHandler>() ->
        Event("SetSum1", &TestBus::Events::SetSum1) ->
        Event("GetSum1", &TestBus::Events::GetSum1) ->
    ;

    -- Lua:
    local result = TestBus.Broadcast.GetSum1(1)

For information on addressing EBuses, see Working with the Event Bus (EBus) system (p. 1851).

For more information about the behavior context, see Behavior Context in Depth (p. 1059).

Behavior Context in Depth

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In Lumberyard 1.8, the behavior context replaces script context. The behavior context works with serialize context, edit context, and network context to provide rich C++ reflection. The behavior context focuses on the runtime aspects of C++ code and allows you to manipulate C++ code and objects while they are being created. All script bindings, including Lua, use this reflection. Reflection is also used for modification of objects while in running state (such as animating object properties) and reading of current properties for component state transitions. You can have multiple behavior contexts that are specialized for different purposes, and you can unreflect the behavior contexts in order to implement reloading. At a high level, the behavior context uses only a few primitives on which to build: properties, methods, classes, EBuses and attributes.

For C++ API reference documentation on the behavior context, see the BehaviorContext Class Reference in the Amazon Lumberyard C++ API Reference.

Reflection API

This section describes how methods, properties, classes, and other primitives are used in the Lumberyard reflection API.
Method

Methods reflect a C++ function. You can have global or class methods. Each method must have a unique name for its scope (global or class). You can also provide default values; this allows you to call reflected methods with fewer arguments. Default values are used right to left. You can also provide a debug description for your method. It's highly recommended that you do so. The debug description information is provided to users for context when they use the reflection, as is done with ClassView in the Lua editor. The system automatically generates a description that includes result and argument types. However, adding intent and additional details to your descriptions greatly improves the usability of your reflection. This is especially important when a method has many attributes.

Property

Properties access data and can be global properties or class properties. Each property must have a unique name for its scope. As is customary, a property has getter and setter methods. If you don’t provide a setter method for a property, the property is read only. If you don’t provide a getter method, the property is write only.

Lumberyard does support global functions, member functions, and Lambda functions as property getters and setters.

Lumberyard provides macros that you can use to wrap a class value. You can use BehaviorValueProperty(&value) to implement getter and setter methods, or you can implement them individually by using BehaviorValueGetter and BehaviorValueSetter. These macros implement Lambda functions for those values. When the state of your object is modified, you might have to perform operations other than simply setting the value. For this reason, it is a best practice to always implement your getters and setters. You can always change your implementation later.

Constant

Constants are implemented as read-only properties and can be global or restricted to a class. A behavior context macro called BehaviorConstant implements the Lambda getter for you.

Enum

Because class enums often require casting, Lumberyard currently treats all enums values as int. Enums are implemented as read-only int properties.

Class

Reflects a C++ class or struct. You can provide an optional name. If you do not provide a name, the class name from AzTypeInfo is used. That name must be unique for the scope. Because the system uses AzRTTI to build the class hierarchy, you can use RTTI if you want to reflect base class functionality.

- **Allocator** - You can provide a custom allocator/deallocator for your class. This allows you to override any existing allocation schema. If you do not provide a custom allocator, aznew/delete is used (AZ_CLASS_ALLOCATOR).
- **Constructor** - Allows you to enumerate the class constructors that you want to reflect. You must pass all constructor arguments as template augments.
- **Wrapping/WrappingMember** - Allows code to inform the system that it is a wrapper of another class. This is useful when you reflect smart pointers and string wrappers.
- **_userdata** - Allows you to provide a pointer to user data. The pointer is accessible from all callbacks (like a custom allocator) that you implement for the class.
- **Method** - Reflects a C++ class function. The first argument is the class pointer. This is the same usage as global methods.
- **Property** - Reflects class data. The first argument is the class pointer. This is the same usage as global properties.
- **Enum** - Enums are int read-only properties.
• **Constant** - Constants are read-only properties.

**Nested Classes**

To bind a nested class to the behavior context, you must do it from within a function on the nested class. C++ rules about nested class member access from outside scopes make this requirement necessary.

The following counterexample shows a poorly formed nested class. The code does not work.

```cpp
//Bad nested class
class Outer
{
    public:
        AZ_TYPE_INFO(Outer, "...");
        static void Reflect(AZ::ReflectContext* context);
        class Inner
        {
            public:
                AZ_TYPE_INFO(Inner, "...");
                int m_member = 0;
        }
};
void Outer::Reflect(AZ::ReflectContext* context)
{
    if (AZ::BehaviorContext* behavior = azrtti_cast<AZ::BehaviorContext*>(context))
    {
        behavior->Class<Inner>("OuterInner")
        ->Property("member", BehaviorValueProperty(&Inner::m_member))
    }
}
```

The following code shows a well formed nested class.

```cpp
//Good nested class
class Outer
{
    public:
        AZ_TYPE_INFO(Outer, "...");
        static void Reflect(AZ::ReflectContext* context);
        class Inner
        {
            public:
                AZ_TYPE_INFO(Inner, "...");
                static void Reflect(AZ::ReflectContext* context);
                int m_member = 0;
        }
};
void Outer::Reflect(AZ::ReflectContext* context)
{
    Inner::Reflect(context);
}
void Outer::Inner::Reflect(AZ::ReflectContext* context)
{
    if (AZ::BehaviorContext* behavior = azrtti_cast<AZ::BehaviorContext*>(context))
    {
```
EBus

EBus reflects Lumberyard event bus messages. Depending on your EBus configuration, Broadcast, Event (with ID) and Queuing are reflected. Queuing is a generic function to be executed when the bus messages are consumed.

- **Event** - Reflects an EBus event. Depending on your EBus configuration, Lumberyard automatically reflects Broadcast, Event, QueueBroadcast, and QueueEvent.
- **Handler** - Reflects a class that you must implement to forward messages from the EBus to behavior context methods. You must create a class that can monitor the specified EBus and forward messages to the behavior context. This is a requirement because the behavior context cannot guarantee that there is a handler for each message. If a message expects a result, you must provide a default result in case the message is not handled by the behavior context user. Keep in mind that the system creates as many of these handlers as the behavior context requires. Handlers can also execute in different threads. As a result, you should avoid static storage for values that change. The best way to understand this is to examine the example that follows.

Example

The following code example shows the use of the Lumberyard reflection API.

```cpp
// Global Property
behaviorContext.Property("globalProperty", &GlobalPropertyGetter, &GlobalPropertySetter); // Property with getter and setter function
behaviorContext.Property("globalProperty", []() { return g_globalValue; }, [](int value) { g_globalValue = value; }); // Property with Lambda functions.
behaviorContext.Property("globalReadOnlyProperty", &GlobalPropertyGetter, nullptr); // Read only property with a getter function.
behaviorContext.Property("globalReadOnlyProperty", []() { return g_globalValue; }, nullptr); // Read only property with a Lambda function.
behaviorContext.Property("globalReadOnlyProperty", BehaviorValueGetter(&g_globalValue), nullptr); // Read only property with a value and a helper macro.
// Write only is the same as ReadOnly, but with the setter enabled and the getter set to nullptr. These properties are rare.

// Global Methods
behaviorContext.Method("GlobalMethod", &GlobalMethod);

// Global Constants and Enums (implemented using properties). The functions are provided for clarity.
behaviorContext.Constant("PI", []() { return 3.14f; });
behaviorContext.Constant("PI", BehaviorConstant(3.14f));
behaviorContext.Enum<EnumIntValue>("EnumIntValue");

// Class - When you declare a class, if you want to reflect base class functionality, just use RTTI.
behaviorContext.Class<MyClass>() // The name of the class comes from AzTypeInfo. In this case the name is "MyClass".
    ->Constructor<int>() // Optional additional constructors. You can have as many as needed.
```
Constant("epsilon",BehaviorConstant(0.001f)) // Class constant. All features from the global versions apply.
Enum<MyClass::ENUM_VALUE>("ENUM_VALUE") // Class enum. All features from the global versions apply.
Method("Method",&MyClass::Method) // Class method. All features from the global versions apply.
Property("data", &MyClass::GetData(), &MyClass::SetData) // Class features. All features from the global versions apply.

// EBus
class MyEBusBehaviorHandler : public MyEBus::Handler, public AZ::BehaviorEBusHandler
{
public:
AZ_EBUS_BEHAVIOR_BINDER(MyEBusBehaviorHandler ,"{19F5C8C8-4260-46B1-B624-997CD3F10CBD}", AZ::SystemAllocator, // Name, TypeId and default allocator.
  OnEvent);// List of event names to handle and support for BehaviorContext.

void OnEvent(int a) override // This is an event listener like other EBus listeners.
{
  Call(FN_OnEvent,a); // Forward the event to a behavior listener if there is one.
  FN_***EventName*** events are declared by the AZ_EBUS_BEHAVIOR_BINDER macro.
}
};

behaviorClass.EBus<MyEBus>("MyEBus") // EBuses are not required to have TypeInfo, so you must always provide a name.
->Handler<MyEBusBehaviorHandler >() // Allow systems that use behavior context to create handlers for this EBus every time
  // they must listen for events. If you reflect a bus without a handler, behavior context users can only send events.
->Event("OnEvent", &MyEBus::Events::OnEvent) // Allow behavior context system to send an "OnEvent" event. The code automatically generates // Broadcast, Event, QueueBroadcast, QueueEvent, and QueueFunctions if the EBus configuration // supports them. You don’t have to provide events; you can only provide a handler if // you don’t have behavior context systems to send events.
;

// Properties, methods, classes and ebuses can have attributes. An attribute is a combination of a Crc32 ID and a value. The value // can be a constant, a variable address, a global function, a class member function, or a class member variable address.
behaviorClass.Method("GlobalMethod", &GlobalMethod)
  ->Attribute("ValueAttr",10) // Value attribute.
  ->Attribute("MethodAttr", &SomeOtherGlobalMethod)
;
// You add the same attributes to a property...
behaviorClass.Property("GlobalProperty", BehaviorValueProperty(&g_globalValue)
  ->Attribute("MyAttr",20)
;
// or to a class or class method or property.
behaviorClass.Class<MyClass>()
  ->Attribute("ClassAttr",100)
  ->Attribute("ClassAttr1", &MyClass::SomeMethod)
  ->Method("Method", &MyClass::Method)
  ->Attribute("MethodAttr",100)
  ->Property("data", BehaviorValueProperty(&MyClass::m_data))
    ->Attribute("PropertyAttr",500)
;
Add modular features and assets with Gems

Gems are packages that contain code and assets to augment your game projects. With the Programming with Gems (p. 1073), you can choose the features and assets that you need for your game project without including unnecessary components. For a list of all gems included in Lumberyard, see Gems Available in Lumberyard (p. 1113).

Lumberyard features two types of Gems:

- **Code & Assets** – Contains assets as well as code that performs certain functions upon the assets.
- **Assets Only** – Contains only assets and no code.

All Lumberyard gems are located in the following directory:

```
lumberyard_version\dev\Gems
```

The gems that you enable are automatically detected and built through the integrated Using the Waf Build System (p. 63).

**Note**

To enable **Code & Assets** Gems, you must select the Create, Modify and Build projects option in Lumberyard Setup Assistant. This option is not required for **Assets Only** gems. For more information, see Running Lumberyard Setup Assistant (p. 16).

You can enable Gems with the Project Configurator (p. 44) or the command line (Lmbr.exe (p. 56)).

**Topics**

- Enabling Gems (p. 1064)
- Creating a Gem (p. 1066)
- Programming with Gems (p. 1073)
- Gems Available in Lumberyard (p. 1113)

---

**Enabling Gems**

You can enable gems using the Project Configurator or from a command line. To enable a gem from a command line, see Gems Commands (p. 58).

**To enable gems with the Project Configurator**

1. Open the Lumberyard Project Configurator, located at `lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe`. For example, when using Visual Studio
2017 as your build platform, the Project Configurator is located at `lumberyard-version\dev\Bin64\vc141\ProjectConfigurator.exe`.

2. In the Project Configurator, select your active game project and choose **Set as default**.

3. Click **Enable Gems**.

4. Select the gems that you want to enable.

5. Click **Save**.

6. If you enabled gems labeled **CODE & ASSETS**, you must build your game project to make the gems available in Lumberyard Editor. If you enabled only gems labeled **ASSETS ONLY**, you do not need to build your game project before opening it in Lumberyard Editor.

For more information, see **Building Lumberyard projects (p. 61)**.
Creating a Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create your own gems by packaging your assets into the Lumberyard gem format. This makes your package distributable, shareable between projects, and easy for game developers to enable or disable as needed.

Creating your gem involves placing code, assets, and several other special files into the correct directory structure. This chapter describes the creating a gem procedure in detail.

To structure your assets as a gem, see the following.

Topics

• Creating a New Empty Gem (p. 1066)
• Packaging Your Assets (p. 1071)
• Testing Your Gem (p. 1072)
• Compressing Your Gem (p. 1072)

Creating a New Empty Gem

You can create a new, empty gem using either the Project Configurator (p. 1066) or the command line (p. 1069). When you create a new gem, Lumberyard automatically creates a directory structure for your gem in the `\dev\Gems\` directory. It also adds the special gem files and directories needed to function as a gem.

Topics

• Creating a Gem with the Project Configurator (p. 1066)
• Creating a Gem from the Command Line (p. 1069)

Creating a Gem with the Project Configurator

Use Project Configurator to create a new, empty gem.

1. Open the Lumberyard Project Configurator, located at `lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe`. For example, when using Visual Studio...
2017 as your build platform, the Project Configurator is located at `lumberyard-version\dev\Bin64vc141\ProjectConfigurator.exe`.

2. Select the project and click **Enable Gems**.
3. Click **Create a new Gem** on the gems page for the game project that you selected.

4. In the **Create a new gem** dialog box:
   a. For **Name**, enter a name for the gem (for example, *MyNewGem*). The name must start with a letter, and only alphanumeric characters and underscore are allowed for the rest of the name. Other special characters or spaces are not allowed in the name.
   b. (Optional) For **Summary**, enter a description for the gem.

      This field also supports hyperlinks using a simple html anchor tag. When entering this directly into the **Summary** field in the Project Configurator, use the following syntax:

      ```html
      <a href="http://www.amazon.com">Amazon</a>
      ```

      If you are editing the Summary parameter in the gem.json file, use the following syntax:

      ```json
      "Summary": "<a href="http://www.amazon.com">Amazon</a>",
      ```
   c. For **Type**, select **Code & Assets** or **Assets Only**.
      - **Code & Assets** – Contains assets as well as code that performs certain functions upon the assets.
      - **Assets Only** – Contains only assets and no code.
   d. Choose **Create Gem**.
5. Close the Project Configurator.

The following directories and files are created in `\dev\Gems\gem_name\` for ‘Assets Only’ and ‘Code & Assets’ gems:

- **Assets** – Directory of assets for your gem, such as materials, models, textures, and audio files. The AZ::IO system automatically includes this directory so that you can reference assets provided by your gem. For example, if you have a material file, `\Assets\materials\MyMaterial.mtl`, you can reference it in your code with the path `materials\MyMaterial.mtl`.

  **Note**
  If this directory is not present, you can manually add it. If you don't have any assets in your gem, you can remove this folder.

- **Gem.json** – Metadata for this gem. Do not modify the **Uuid** or **Name** fields. You can specify the following gem metadata fields:
  
  - **Version** – `n.n.n` (numerical values only).
    
    Gem revision is not supported, so you should not plan to increment the revision after shipping the gem.
  
  - **DisplayName** – Friendly display name. Can contain spaces.
  
  - **Tags** – Searchable tags that are displayed in Project Configurator under the gem’s summary. Enter this as a comma-separated list of strings.
  
  - **IconPath** – Path to the gem’s display icon.
  
  - **Summary** – Detailed description of the gem. This parameter supports hyperlinks. To specify a hyperlink, use the following syntax:

    ```json
    "Summary": "<a href="http://www.amazon.com">Amazon</a>",
    ```

  - **Dependencies** – You can specify a dependency on another gem or on an engine version. Edit the `_comment` metadata.

    ```json
    "Dependencies": [
      {
        "Uuid": "540fa970c994668b5d02c66a39c6625",
        "VersionConstraints": [">1"
        ],
        "_comment": "zzzTestVer001"
      }
    ]
    ```
Creating a New Empty Gem

- preview.png – Preview image displayed in the Project Configurator gem's list.

For 'Code & Assets' gems, the following files and directories are also created (in addition to the files listed above):

- Code\gem_name\waf_files – Waf files json for your gem that specifies which files are built, how files should be combined into uber files, and how files are filtered in the Visual Studio Project.
- Code\gem_name\_test.waf_files – Additional files to include when building with the test specification.
- Code\wscript – Waf wscript Python file that defines libraries and other build settings used by your gem.
- Code\Include\gem_name\gem_name\Bus.h – Globally visible header that can be included by any project that uses your gem, or by other gems that depend on it. External code can call into your gem's module and receive events from your module through public event buses. Event buses allow simple and safe function calls between different modules of code.
- Code\Source\gem_name\Module.cpp – Implementation of the gem_name Module. Starting in Lumberyard 1.5, new gems are built around AZ modules. The default module registers one system component, the gem_name\SystemComponent.
- Code\Source\gem_name\SystemComponent.h – Header for the gem_name\SystemComponent implementation. This component is registered by the gem_name\Module and is a handler for the gem_name\Bus.
- Code\Source\gem_name\SystemComponent.cpp – gem_name\SystemComponent implementation. Inside this component are the typical Activate, Deactivate, Init, and Reflect methods.
- Code\Source\StdAfx.cpp – Code\Source\StdAfx.h
- Code\Tests\gem_name\Test.cpp

6. Build your game project:

- If your gem is a Code & Assets gem, you must build your project (p. 61).
- If your gem is an Assets Only gem, you do not need to build your project.

Creating a Gem from the Command Line

You can also use the command line to create a gem. You can do this in either the command line or the Project Configurator (p. 1066).

If you add a Code & Assets gem, you must build your project (p. 61) after adding the gem. Asset Only gems do not require a rebuild.

To create a gem with the command line

1. Open a command-line prompt and navigate to the lumberyard_version\dev\Bin64\vc141 directory.
2. Enter one of the following commands:

   - Create a Code & Asset gem if your gem contains assets as well as code that performs certain functions upon the assets.
   - lmbr gems create MyNewGem

   - Create an Assets Only gem if your gem contains only assets and no code.
Creating a New Empty Gem

```
lmbr gems create MyNewGem -asset-only
```

3. Close the command line.

The following directories and files are created in `\dev\gems\gem_name\` for 'Assets Only' and 'Code & Assets' gems:

- **Assets** – Directory of assets for your gem, such as materials, models, textures, and audio files. The AZ::IO system automatically includes this directory so that you can reference assets provided by your gem. For example, if you have a material file, `\Assets\materials\MyMaterial.mtl`, you can reference it in your code with the path `materials\MyMaterial.mtl`.

  **Note**
  If this directory isn't present, you can manually add it. If you don't have any assets in your gem, you can remove this folder.

- **Gem.json** – Metadata for this gem. Don't modify the Uuid or Name fields. You can specify the following gem metadata fields:
  - **Version** – n.n.n (numerical values only).
    Gem revision isn't supported, so you shouldn't plan to increment the revision after shipping the gem.
  - **DisplayName** – Friendly display name. Can contain spaces.
  - **Tags** – Searchable tags that are displayed in Project Configurator under the gem's summary. Enter this as a comma-separated list of strings.
  - **IconPath** – Path to the gem's display icon.
  - **Summary** – Detailed description of the gem. This parameter supports hyperlinks. To specify a hyperlink, use the following syntax:
    ```
    "Summary": "<a href="http:\/\/www.amazon.com\">Amazon</a>",
    ```
  - **Dependencies** – You can specify a dependency on another gem or on an engine version. Edit the `_comment` metadata.
    ```
    "Dependencies": [
      {
        "Uuid": "540faf970c994668b5d02c66a39c6625",
        "VersionConstraints": [
          ">1"
        ],
        "_comment": "zzzTestVer001"
      }
    ],
    ```
  - **preview.png** – Preview image displayed in the Project Configurator gem's list.

For 'Code & Assets' gems, the following files and directories are also created (in addition to the files listed above):

- **Code\gem_name.waf_files** – Waf files json for your gem that specifies which files are built, how files should be combined into uber files, and how files are filtered in the Visual Studio Project.
- **Code\gem_name_test.waf_files** – Additional files to include when building with the test specification.
- **Code\wscript** – Waf wscript Python file that defines libraries and other build settings used by your gem.
Packaging Your Assets

Before you can include your assets in your new gem, you must place them within your Lumberyard's game project root directory. This way, they will be imported by Lumberyard Asset Pipeline and converted into assets that Lumberyard can use. After you move and import your assets, you must test them in Lumberyard to ensure that they still work as intended. You then move the verified assets into the gem directory structure that was created when you created a new gem.

Tip
To avoid conflicts with similarly named assets, we recommend that you use a namespace workflow or acronyms when you create directories that contain assets in gems. For example, you can add a directory between Assets and Textures so that the directory doesn't conflict with other gems that might have identically named textures. Your structure might look like the following:

\lumberyard_version\dev\Gems\gem_name\Assets\example_name\Textures\CommonTextureName.tif

To package your assets

1. Prepare your asset directory.
   a. Create a directory directly beneath your game project's root directory. For example, the Samples Project's game project root directory is \dev\SamplesProject.
   b. Place into the new directory your assets and any related files and subdirectories containing files that your assets use, such as textures. Ensure that you maintain the original directory structure within the asset directory to avoid breaking relative file paths in your assets.
   c. If preferred, you can create additional subdirectories for organization purposes. Be sure to amend any file references that may be broken by the new structure.

2. Import the assets into Lumberyard. To do this, launch Lumberyard. The Asset Processor automatically detects the new assets and imports them.

3. Verify and test the assets.
   a. In Lumberyard, open a level and find your asset directory in the Asset Browser. If the Asset Browser is not open, click Tools, Asset Browser to open it.
b. Drag your assets into the level and verify that they look and function as intended. If necessary, make changes to the assets, such as assigning textures or changing materials.

c. Save any changes that you made to your assets and close Lumberyard.

4. Move the contents of your asset's root directory that you created in Step 1. Be sure to include the metadata files generated by the Asset Processor. Place the contents into the new gem's Assets directory. This is located at `\dev\Gems\gem_name\Assets`.

   **Note**
   The Gem's Assets directory maps to the root of your project folder. For example, if your asset is in `\dev\project_name\my_asset\`, then it maps to the gem at `\dev\Gems\my_gem\Assets\my_asset\`.

### Testing Your Gem

After you move your assets into the gem's directory structure, you can enable your new gem in your game project and test it in Lumberyard.

**To enable and test your new gem**

1. Open the Project Configurator by doing one of the following:
   - Open Lumberyard Setup Assistant and on the **Summary** page, click **Configure project**.
   - Open the Lumberyard Project Configurator directly from `lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe`. For example, when using Visual Studio 2017 as your build platform, the Project Configurator is located at `lumberyard-version\dev\Bin64\vc141\ProjectConfigurator.exe`.

2. Enable (p. 1064) your new gem.

3. Close the **Project Configurator**.

4. Launch Lumberyard and open a level.

5. Use the **Asset Browser** to place the gem's assets into your scene. Verify that your assets look as intended. If they don't, verify that you have set up the directories correctly.

### Compressing Your Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can compress your gem to make it distributable to other Lumberyard game developers. To do this, you zip the contents of the gem directory using your preferred zip tool, and change the `.zip` filename extension to `.crate`. The compressed `.crate` file must be under 4GB in size. By renaming the `.zip` to `.crate`, game developers who use your gem can unpack and install it by drag and dropping it into Lumberyard.

**To compress your gem into a crate file**

1. Using a file browser, open your gem's directory. For example, `lumberyard_version\dev\Gems\my_gem`.

2. Select the contents of this directory and use your preferred zip tool to zip them.

   **Note**
   Don't include your gem's top level directory as part of the zip file. Select the files and directories inside the `my_gem` directory to zip.
3. After the .zip file is created, remove the .zip extension and rename it to .crate.

Your gem crate is ready for distribution.

Programming with Gems

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The modular gems system is a management infrastructure for sharing code and art assets between Lumberyard game projects. The modular gems system consists of gem packages that you can access and manage with the Project Configurator (p. 43) or \lmbx.exe (p. 56). You can use gems to add functionality such as code, art, scripts, supporting files, and references to other gems to your game project.

Topics
- Gem Structure (p. 1073)
- Adding C++ Code to a Lumberyard Game with Gems (p. 1086)
- Using AZ Modules to Initialize Gems (p. 1090)

Gem Structure

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Each gem package can contain the following:
• Assets
  • A `gem.json` file
  • An icon file
  • Code

To locate Lumberyard Gems

• Navigate to the `lumberyard_version\dev\Gems` directory. For more information, see Gems Available in Lumberyard (p. 1113).

Topics

• Gem Assets (p. 1074)
• Gem Icon File (p. 1074)
• Gem JSON File (p. 1074)
• Gem Code (p. 1077)
• Gem List File – `gems.json` (p. 1079)
• Accessing Gems in Code (p. 1079)
• Gem Configuration Examples (p. 1080)

Gem Assets

Each gem has an `Assets` directory that contains models, textures, scripts, and animations. Asset files are accessed the same way as they are in a game project. Lumberyard uses this root directory to find the asset file path. For example, when Lumberyard looks for the `textures\rain\rainfall_ddn.tif` file, it looks in the `<GemName>\Assets\textures\rain\rainfall_ddn.tif` directory.

Gem Icon File

The gem icon is a `.jpg`, `.png`, or `.gif` file. The following example is the icon for the Lightning Arc Gem (p. 1152).

![Lightning Arc Gem Icon](image)

Gem JSON File

Example

This example file contains metadata for the gem and the following properties.

```json
{
  "GemFormatVersion": 4, // Required
  "Uuid": "e5f049ad7f534847a89c27339cf6a6", // Required
  "Name": "ExampleGem", // Required
  "DisplayName": "Environment Gem", // Optional, defaults to the [Name] field
  "Summary": "Gem description.", // Optional, defaults to ""
}```
"Tags": [  "Weather",  "Weather Effects",  "Sky" ],
"EditorTargets": ["RainEditorPlugin"],
"LumberyardVersion": ["==1.11.0.0"],
"Version": "1.0.0",
"IconPath": "path\preview.png",
"Modules": [
  {
    "Type": "ModuleType",
    "Name": "ModuleName",
    "Extends": "GameModule"
  }
],

"Dependencies": [
  {
    "Uuid": "d378b5a7b47747d0a7aa741945df58f3",
    "VersionConstraints": ["~>1.0"],
    
  }
]

GemFormatVersion

Version number that identifies how a gem is built (folder structure and semantics) and determines compatibility with specific versions of Lumberyard. The version number is set automatically when you create a gem in the Project Configurator.

- Lumberyard 1.11 and later – Gems have a default value of 4 for the GemFormatVersion field.
- Lumberyard 1.5 to 1.10 – Gems have a default value of 3 for the GemFormatVersion field. The gems that are included with Lumberyard are AZ modules.
- Lumberyard 1.4 and earlier – Gems have a default value of 2 for the GemFormatVersion field.

Uuid

Unique ID that identifies the gem.

Name

Name of the gem.

DisplayName

Friendly name of the gem that appears in the Project Configurator.

Summary

Short description of the gem.

Tags

List of tags that describe the gem.

EditorTargets

List of additional Waf targets to build. This is useful for targets defined in the wscript such as adding editor plugins that accompany gems.

LumberyardVersion

Version of Lumberyard to specify. The value can be one of the following:
- An array of strings (for example, ["~>1.2.345.6789", "~>1.1.1", ">= 1.0"]).
• A single string that describes the constraint (for example, "==1.2.3").

For more examples of acceptable version specifiers, see the list under Dependencies (p. 1076).

**Version**

API version of the gem. The version should follow the Semantic Versioning specification.

**IconPath**

Path from the gem directory to the display icon. The icon can be a .jpg, .png, or .gif file. The recommended icon size is 160 x 90 px.

**Modules**

Code product that the gem produces. Each module entry can have the following fields:

• **Type** – Type of module that is defined. See Module Types (p. 1077).

• **Name** – Name of the module. You use this name when you refer to the module, and in the module's output file name.

• **Extends** – Module to use as a base. The value can be only "GameModule" and you can use this field only for the following module types:
  • EditorModule
  • Builder

If the module does not have a name, specify "GameModule".

By default, this module takes the following fields from the extended module's build settings. If you specify any of the following values, they override the parent module.

• file_list
• platforms
• configurations
• defines
• includes
• features
• lib
• libpath
• use
• uselib

**Dependencies**

UUIDs and versions of other gems to which the gem depends. Acceptable version specifiers have an operator and a version number. Consider these examples:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>== 1.2.3</td>
<td>Minimum: 1.2.3</td>
<td>Maximum: 1.2.3</td>
</tr>
<tr>
<td>&gt;= 1.2.3</td>
<td>Minimum: 1.2.3</td>
<td>Maximum: None</td>
</tr>
<tr>
<td>&lt;= 1.2.3</td>
<td>Minimum: None</td>
<td>Maximum: 1.2.3</td>
</tr>
<tr>
<td>&gt; 2.0.0</td>
<td>Minimum: 2.0.0 (exclusive)</td>
<td>Maximum: None</td>
</tr>
<tr>
<td>&lt; 2.0.0</td>
<td>Minimum: None</td>
<td>Maximum: 2.0.0 (exclusive)</td>
</tr>
<tr>
<td>~&gt; 1.2.3</td>
<td>Minimum: 1.2.3</td>
<td>Maximum: 1.3.0 (exclusive)</td>
</tr>
<tr>
<td>-&gt; 1.2</td>
<td>Minimum: 1.2.0</td>
<td>Maximum: 2.0.0 (exclusive)</td>
</tr>
</tbody>
</table>
The following fields apply to GemFormatVersion 3 and earlier:

**EditorModule**

Defines whether the gem produces a module that is loaded by Lumberyard Editor.

**LinkType**

Defines how other gems and game projects should link to this gem:
- **Static** – Does not produce a .dll file and does not link.
- **Dynamic** – Produces a .dll file and links all dependent projects against the .dll file using an import library.
- **NoCode** – Does not produce a .dll or .lib file. The gem contains assets but not code. You can add a gem with the NoCode value to a game project without rebuilding.

**Module Types**

Following are module types and how to define them. For examples of gems with module types defined, see Gem Configuration Examples (p. 1080).

**GameModule**

The most basic type of module. If you used gems before, your gems produce this type of module by default. These modules are loaded by Lumberyard Setup Assistant and Lumberyard Editor.

*Note*

If a gem has more than one GameModule type, you can omit the Name field for one GameModule type. This allows a gem's product .dll file to keep the same name when you upgrade to GemFormatVersion 4.

**EditorModule**

Loaded by Lumberyard Editor. If an EditorModule extends a GameModule, the EditorModule is loaded instead of the GameModule in Lumberyard Editor. This is useful when an EditorModule contains the same components as a GameModule, along with their editor components.

**StaticLib**

Module type that is useful when your gem requires helper code that is linked to all dependents of your gem, or when your gem provides only statically usable code. All modules in all gems that require this gem have this StaticLib linked against it.

**Builder**

Used by Asset Processor for building assets. For more information, see Creating a Custom Asset Builder (p. 318).

**Gem Code**

You can find a gem's code in the `<GemName>\Code` directory. The directory contains the following:

**wscript**

Waf build script that is generated by the template. The script contains all build configuration options, including target name, included paths, required libs, defines, and so on. In most cases, this script does not need to be changed.
For more information, see Wscript and Modules (p. 1078).

**gemname.waf_files**

JSON list of all files that are included in the game project. The root object contains properties for each uber file, and a special `NoUberFile` object. Each child object contains a named array of files, where the name is the filter that is used in generated projects. The gem template provides a default `.waf_files` list. All new files for your game project should be added to this file.

**gemname_tests.waf_files**

JSON list of all test files for a gem, in the same format as `gemname.waf_files`.

**Include/GemName**

Directory contains headers that define the gem's public API.

Other gems can include this directory. It should not contain implementations nor anything other than virtual function definitions. The gem template provides a default `GemNameBus.h` that contains a `GemNameRequestBus` interface, which defines public functionality.

For more information, see Working with the Event Bus (EBus) system (p. 1851).

**Source**

Directory that contains the following generated files:

- `StdAfx.h` – Includes frequently required files.
- `StdAfx.cpp` – Includes the `StdAfx.h` file.
- `GemNameSystemComponent.h` – Contains the definition of a system component that handles calls to `GemNameRequestBus`.
- `GemNameSystemComponent.cpp` – Contains the implementation of the `GemNameSystemComponent` class.
- `GemNameModule.cpp` – Contains the `AZ::Module` class definition, which is used to register components and do additional component reflection.

**Note**

This class can be made to extend `CryHooksModule` (in `IGem.h`) instead to have `gEnv` attached automatically.

**Tests**

Directory that contains unit tests for your gem. Add all files in this directory to the `gemname_tests.waf_files`.

- `GemNameTest.cpp` – Contains an example for writing gtests for your gem, which are unit test files for your gem.

**Wscript and Modules**

The gem's `wscript` must call `bld.DefineGem(...)` in order for the gem to build properly. The `DefineGem` function expects keyword arguments, mapping from a module's name to a data type dictionary of the build settings for the module. If the module does not have a name, its build settings may be placed directly in the call to `DefineGem()`.

For each module, the following default settings are applied:

- **target**: `gem_name_module_name`

  If the module does not have a name, the default value is `gem_name`.

- **file_list**: `gem_name_module_name.waf_files`

  If the module does not have a name, the default value is `gem_name.waf_files`
• **test file_list**: `gem_name_module_name_tests.waf_files`  
  If this file exists, it is automatically added to the `file_list` when building tests.

• **vs_filter**: "Gems"

• **output_file_name**: `Gem.gem_name.module_name.gem_uuid.gem_version`  
  If the module does not have a name, the default value is `Gem.gem_name.gem_uuid.gem_version`.

• **platforms**: all  
  For Builder and EditorModule module types, the default value is: ['win', 'darwin'].

  For more information, see *Module Types* (p. 1077).

  **Note**  
  Overriding this value is not recommended.

• **configurations**: all  
  For Builder and EditorModule module types, the default value is: ['debug_all', 'profile_all'].

  **Note**  
  Overriding this value is not recommended.

• **includes**: ['Include', 'Source']  
  This also contains the include path of all dependencies.

• **pch**: 'StdAfx.cpp'  
  **use**: All modules that require linking from all dependencies.

For more information about waf and wscript files, see *Using the Waf Build System* (p. 63).

### Gem List File – gems.json

You can find the `gems.json` list file at the root of each project directory. For example, for the Samples Project, the `gems.json` file is located in the `lumberyard_version\dev\SamplesProject` directory. This example file lists all gems used in the game project.

**Example**

```json
{
    "GemListFormatVersion": 2,  // Required
    "Gems": [  
        {
            "Path": "Gems/Rain",  // Required
            "Uuid": "e5f049ad7f534847a89c27b7339c6f6a6",  // Required
            "Version": "0.1.0",  // Required
            "_comment": "Rain"  // Optional, useful comment
        }
    ]
}
```

### Accessing Gems in Code

You can access gems through code, as in the following example:

```cpp
#include <GemName/GemNameBus.h>

EBUS_EVENT(GemName::GemNameRequestBus, MyFunction, withArgs);
```
Gem Configuration Examples

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following are example configurations for gems and their wscript files.

Contents

- Gem with a GameModule (p. 1080)
- Gem with a GameModule and Name (p. 1080)
- Gem with a StaticLib and Name (p. 1081)
- Gem with an EditorModule and Name (p. 1081)
- Gem with an EditorModule that Extends a GameModule (p. 1082)
- Gem with a StaticLib and Standalone Module (p. 1083)
- Gem with a Header Only Library with Unit Testing (p. 1084)
- Gem with a StaticLib Only and Unit Testing (p. 1085)

Gem with a GameModule

The following example gem adds one module type as GameModule.

```json
{  
  "GemFormatVersion": 4,
  "Uuid": "e5f049ad7f534847a89c27b7339cf6a6", // Required
  "Name": "MyGem", // Required
  "Version": "1.0.0", // Required
  "Modules": [  // Optional
    
    {  
      "Type": "GameModule"  // Required
    }
  ]
}
```

The following is the wscript file for the gem.

```python
def build(bld):
    bld.DefineGem(
        # The following are the default settings, and do not need to be specified (unless changes are preferred)
        # target    = ['MyGem'],
        # file_list = ['mygem.waf_files'],
        # vs_filter = ['Gems'],
    )
```

Gem with a GameModule and Name

The following example gem has one module type as GameModule and specifies a name for that module.

```json
{  
  "GemFormatVersion": 4, // Required
  "Uuid": "e5f049ad7f534847a89c27b7339cf6a6", // Required
  "Name": "MyGem", // Required
```

Version 1.28
1080
The following is the `wscript` file for the gem.

```python
def build(bld):
    bld.DefineGem(
        Components = dict(
            # The following are the default settings, and do not need to be specified
            # target    = ['MyGem.Components'],
            # file_list = ['mygem_components.waf_files'],
            # vs_filter = ['Gems'],
            ),
        )
```

**Gem with a StaticLib and Name**

The following example gem has one module type as `StaticLib` and specifies a name for that module.

```python
{
    "GemFormatVersion": 4,                       // Required
    "Uuid": "e5f049ad7f534847a89c27b7339cf6a6", // Required
    "Name": "MyGem",                            // Required
    "Version": "1.0.0",                         // Required
    "Modules": [                                // Optional
        {
            "Type": "StaticLib",                // Required
            "Name": "HelperCode"                // Required
        }
    ]
}
```

The following is the `wscript` file for the gem.

```python
def build(bld):
    bld.DefineGem(
        HelperCode = dict(
            # The following are the default settings, and do not need to be specified
            # target    = ['MyGem.HelperCode'],
            # file_list = ['mygem_helpercode.waf_files'],
            # vs_filter = ['Gems'],
            ),
        )
```

**Gem with an EditorModule and Name**

The following example gem has one module type as `EditorModule` and specifies a name for the module.

```python
{
```
"GemFormatVersion": 4,                      // Required
"Uuid": "e5f049ad7f534f47a89c27b7339cf6a6", // Required
"Name": "MyGem",                            // Required
"Version": "1.0.0",
"Modules": []                                // Optional
{
    "Type": "EditorModule",               // Required
    "Name": "Editor",                  // Required
}

} 

The following is the wscript file for the gem.

def build(bld):
    bld.DefineGem(
        Editor = dict(
            # The following are the default settings, and do not need to be specified
            # unless changes are preferred
            # target    = ['MyGem.Editor'],
            # file_list = ['mygem_editor.waf_files'],
            # vs_filter = ['Gems'],
            # platforms = ['win', 'darwin'],
            # configurations = ['debug', 'debug_test', 'profile', 'profile_test'],
        ),
    )


Gem with an EditorModule that Extends a GameModule

The following example gem has two module types, GameModule and EditorModule, specifies a name for both, and then extends the EditorModule.

{
    "GemFormatVersion": 4,                      // Required
    "Uuid": "e5f049ad7f534f47a89c27b7339cf6a6", // Required
    "Name": "MyGem",                            // Required
    "Version": "1.0.0",
    "Modules": []                                // Optional
    {
        "Type": "GameModule",               // Required
        "Name": "Components"                // Optional
    },
    {
        "Type": "EditorModule",            // Required
        "Name": "Editor",                  // Required
        "Extends": "Components"            // Optional
    }
}

The following is the wscript file for the gem.

def build(bld):
    bld.DefineGem(
        Components = dict(
            # The following are the default settings, and do not need to be specified
            # unless changes are preferred
            # target    = ['MyGem.Components'],
            # file_list = ['mygem_components.waf_files'],
        ),
    )
Gem Structure

# vs_filter = ['Gems'],
defines   = ['MY_GEM'],
use        = ['AzCore'],
),
Editor = dict(
# The following are the default settings, and do not need to be specified
# (unless changes are preferred)
# target    = ['MyGem.Editor'],
# file_list = ['mygem_components.waf_files', 'mygem_editor.waf_files'],
# vs_filter = ['Gems'],
# platforms = ['win', 'darwin'],
# configurations = ['debug', 'debug_test', 'profile', 'profile_test'],
# defines   = ['MY_GEM'], # this is inherited from Components
# this is inherited from Components
use        = ['AzToolsFramework'], # This overrides the Components module's
    'use', so only 'AzToolsFramework' is 'use'd
   ),
}

Gem with a StaticLib and Standalone Module

The following example gem has a StaticLib module, a Standalone module, and specifies a name for both.

{
    "GemFormatVersion": 4,
    "Uuid": "089562a2cbbd41749b359f85fa04f1c9",
    "Name": "CrashReporting",
    "DisplayName": "CrashReporting",
    "Version": "0.1.0",
    "Summary": "Enable external crash reporting for a game project",
    "Tags": ["Untagged"],
    "IconPath": "preview.png",
    "Modules": [
        {
            "Type": "StaticLib",
            "Name": "StaticLibrary"
        },
        {
            "Type": "Standalone",
            "Name": "Uploader"
        }
    ]
}

The following is the wscript file for the gem.

```python
def build(bld):
    bld.DefineGem(
        StaticLibrary = dict(
            platforms   = ['win'],
            includes    = ['.', 'Include'],
            file_list   = ['crashreporting_static.waf_files'],
            win_file_list = ['crashreporting_static_win.waf_files'],
            use         = ['CrashHandler']
        )
    )

    uploader_file_list = ['game_crash_uploader.waf_files']
    platform = bld.env['PLATFORM']
    if platform == 'project_generator':
```

Version 1.28

1083
uploader_file_list.append('game_crash_uploader_win.waf_files')
uploader_file_list.append('game_crash_uploader_unimplemented.waf_files')
elif platform in ['win_x64_vs2017', 'win_x64_vs2019']:
    uploader_file_list.append('game_crash_uploader_win.waf_files')
else:
    uploader_file_list.append('game_crash_uploader_unimplemented.waf_files')

bld.CryQtApplication(
    # Settings
    target          = 'CrashReporting.Uploader',
    output_file_name = 'GameCrashUploader',
    file_list       = uploader_file_list,
    vs_filter       = 'Gems',
    includes        = ['Include'],
    platforms       = ['win'],
    configurations  = ['all'],
    use             = ['CrashUploaderSupport', 'AzCore'],
    disable_pch     = True,
    win_lib         = ['rpcrt4', 'version', 'powrprof', 'winhttp', 'user32', 'advapi32'])

Gem with a Header Only Library with Unit Testing

The following example gem has a Standalone module with unit testing.

```json
{
    "GemFormatVersion": 4,                      // Required
    "Uuid": "e5f049ad7f534847a89c27b7339cf6a6", // Required
    "Name": "MyGem",                            // Required
    "Version": "1.0.0",                         // Required
    "Modules": [],                              // Optional
    
    
    "Type": "Standalone",               // Required
    "Name": "Lib"                       // Optional
}
```

The following is the wscript file for the gem.

```python
def build(bld):
    bld.CryEngineModule(
        # The following are the default settings, and do not need to be specified
        # (unless changes are preferred)
        # target          = ['MyGem.Lib'],
        # file_list       = ['mygem_lib.waf_files'],
        # vs_filter       = ['Gems'],
        test_all_file_list = ['mygem_lib_tests.waf_files'],
        platforms       = ['win', 'darwin'],
        configurations  = ['debug_test', 'profile_test'],
    )
```

Note
The gem must exist in the gems.json for your project but should be removed from the game.xml and editor.xml files.

1. In Project Configurator, create the gem and make the changes to the gem.json and wscript files.
2. Do one of the following:
Gem with a StaticLib Only and Unit Testing

The following example gem has a StaticLib module, and a Standalone module for unit testing.

```json
{
    "GemFormatVersion": 4, // Required
    "Uuid": "e5f049ad7f534847a89c27b7339cf6a6", // Required
    "Name": "MyGem", // Required
    "Version": "0.1.0",
    "Modules": [
        {
            "Name": "StaticLibrary",
            "Type": "StaticLib"
        },
        {
            "Name": "UnitTestModule",
            "Type": "Standalone",
            "Extends": "StaticLibrary"
        }
    ]
}
```

The following is the wscript file for the gem.

```python
def build(bld):
    bld.DefineGem(
        StaticLibrary = dict(
            # The following are the default settings, and do not need to be specified
            # (unless changes are preferred)
            # target    = ['MyGem.Lib'],
            # file_list = ['mygem_lib.waf_files'],
            # vs_filter = ['Gems'],
        )
    )

    bld.CryEngineModule(
        # The following are the default settings, and do not need to be specified (unless changes are preferred)
        # vs_filter = ['Gems'],
        target          = 'TestGem.UnitTestModule',
        output_file_name = 'TestGemUnitTests',
        file_list = ['mygem_tests.waf_files'],
        platforms   = ['win', 'darwin'],
        configurations = ['debug_test', 'profile_test']
    )
```

**Note**

The gem must exist in the gems.json for your project but should be removed from the game.xml and editor.xml files.

1. In Project Configurator, create the gem and make the changes to the gem.json and wscript files.
2. Do one of the following:
   a. In a command line window, navigate to the \lmbr.exe file and enter the following command.

   \lmbr \projects \populate-appdescriptors

   b. In a text editor, delete the entries for the gem from the game.xml and gem.xml files.

**Adding C++ Code to a Lumberyard Game with Gems**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To add C++ code or assets to your Lumberyard game, use a gem (p. 1064). This document shows you how to create a gem and the various techniques that you can use to add code to it.

**Creating and Configuring Gems**

Creating a gem is straightforward. To create or enable a gem for your game project, follow the steps in the Add modular features and assets with Gems (p. 1064). The gem that you create or enable is located in the \dev\Gems\<gem name> directory. The gem’s directory structure is similar to the following example. Your gem might not have all the directories listed.

```
\3rdParty
\Assets
\Code
 \Include
 \Source
 \Tests
\External
```

- 3rdParty – Contains third-party definition files that are specific to the gem.
- Assets – Contains assets to include in the game project that are specific to the gem.
- Code\Include – An additional header include path to add to a game project that uses the gem.
- Code\Source – Contains the source code for the gem.
- Code\Tests – Contains unit testing source code for projects that are built in a test configuration.
- External – Contains external libraries on which the gem depends that are specific to the gem.

**Code Directory Contents**

The \dev\Gems\<gem name>\Code subdirectory has the following items that Lumberyard creates by default:

<table>
<thead>
<tr>
<th>Item</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>An empty EBus (p. 1851) include file that is named after the gem.</td>
<td>\Include&lt;gem name&gt;&lt;gem name&gt;Bus.h</td>
</tr>
<tr>
<td>The AZ module for the gem</td>
<td>\Source&lt;gem name&gt;Module.cpp</td>
</tr>
</tbody>
</table>
Lumberyard User Guide
Adding C++ Code to a Lumberyard Game with Gems

<table>
<thead>
<tr>
<th>Item</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>The default systems component files</td>
<td>\Source&lt;gem name&gt;SystemComponent.h</td>
</tr>
<tr>
<td></td>
<td>\Source&lt;gem name&gt;SystemComponent.cpp</td>
</tr>
<tr>
<td>Standard precompiled header files</td>
<td>\Source\StdAfx.h</td>
</tr>
<tr>
<td></td>
<td>\Source\StdAfx.cpp</td>
</tr>
<tr>
<td>Skeleton unit test source file</td>
<td>\Tests&lt;gem name&gt;Test.cpp</td>
</tr>
</tbody>
</table>

Lumberyard also creates certain Waf (p. 63)–related files in the \dev\Gems\gem name\Code subdirectory. These files specify the source code content and how the gem is defined and built.

<table>
<thead>
<tr>
<th>Item</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manifest file for the Waf build of the gem</td>
<td>&lt;gem name&gt;.waf_files</td>
</tr>
<tr>
<td>Manifest file that lists additional files to for test configuration builds (for example, Tests/&lt;gem name&gt;Test.cpp)</td>
<td>&lt;gem name&gt;_tests.waf_files</td>
</tr>
<tr>
<td>Waf build script file that defines the gem</td>
<td>wscript</td>
</tr>
</tbody>
</table>

The manifest file that manages the source code uses the Lumberyard Waf waf_files schema to define the source files, their Microsoft Visual Studio filter, and uber (p. 139) file grouping. The default <gem name>.waf_files that the Project Configurator generates looks like the following:

```json
{
    "none": {
        "Source": [
            "\Source/StdAfx.cpp",
            "\Source/StdAfx.h"
        ]
    },
    "auto": {
        "Include": ["\Include/<gem name>/<gem name>Bus.h"
        ],
        "Source": [
            "\Source/<gem name>Module.cpp",
            "\Source/<gem name>SystemComponent.cpp",
            "\Source/<gem name>SystemComponent.h"
        ]
    }
}
```

The .waf_files file contains the following three levels of indentation:

- The first level contains the uber file mapping for uber file–enabled builds.
- none specifies files to be excluded from the uber file. Precompiled headers must be listed here.
- auto specifies files that are automatically combined into modules that are optimized for compile time by Waf.
- You can also specify a fixed uber file name (for example, my_gem_uber_0.cpp) at this level to specially group a set of files. All of the files in the grouping are combined into
my_gem_uber_0.cpp. This technique is useful to restrict which files are combined, like operating system-specific code.

- The second level represents the Visual Studio filters that determine how the files are organized in the Visual Studio solution that Waf generates.
- A special value called Root represents the root node of the project explorer in Microsoft Visual Studio (that is, the file is not placed in any subdirectory).
- The third level contains paths to the source files relative to the location of the waf_files file itself.

### Updating Gem Code

To update code in your gem, add a third-party library, new source code files, system components, or dependencies on other gems or Lumberyard modules.

#### Adding an External Third-Party Library to a Gem

A gem can use any Lumberyard third-party library, including its own private third-party library. To register a private third-party library for a gem, create a third-party configuration file in the \3rdParty directory of the gem. For more information, see Adding Third-Party Libraries (p. 105).

When you set the configuration file to an external third-party library, the library can be added as a gem dependency through the Waf dependency mechanism. For information about third-party library configuration files, see Creating Third-Party Library Configuration Files for Waf (p. 108).

Waf uses two conventions to add dependencies to third-party libraries: uselib and use. The convention uselib is a wscript keyword that configures a module to link to a library's include path, library path, and library. If a gem uses uselib to consume its gem-specific library, then that library is available to the gem only for compiling and linking.

The convention use is similar to the uselib keyword, except that the library's dependencies are recursively propagated to the module that adds the dependency. If a gem uses use to consume its gem-specific library, then the library can be used recursively. This means that if the gem is enabled, the gem-specific third-party library is also available to the game project or dependent gems.

#### Adding New Source C++ Files to Gems

To add source code (for example, C++ or Qt) to a gem, use the Lumberyard Using the Waf Build System (p. 63). Place internal source files (that is, files not meant to be exposed outside of the gem) under the \Code\Source directory. Place header files that can be included in projects or other gems in the \Code\Include\<gem name>\ directory. Place additional unit tests in the \Code\Tests directory.

To add code files to a gem

1. Add the files to the target location. For consistency, we recommend that you place the source files somewhere in the \Code directory.
2. Add the source files to the Waf manifest waf_files file. Following the format described earlier, add the source file paths to either the <gem name>.waf_files file and/or the <gem name>_test.waf_files file.
3. From a command prompt window, run lmbr_waf configure to configure Waf and regenerate the Microsoft Visual Studio solution.

#### Adding System Components

New gems come with a default system component called <gem name>SystemComponent. You can modify this system component according to your requirements. To communicate with the system component, you can define as many EBuses as required. If you want to add additional components,
you must add the component descriptor to the AZ module for the gem (`\Code\Source\<gem
name>Module.cpp`). For more information, see System Components (p. 1098) and Creating System Components (p. 987).

### Adding Dependencies to a Gem

In addition to adding dependencies to gem-specific third-party libraries, your gem can specify dependencies on Lumberyard Engine modules, other gems, or third-party libraries for Lumberyard.

#### Adding a Dependency on a Lumberyard Module

Any gem can be configured to depend on any of the following Lumberyard framework modules:

- AzCore
- AzFramework
- AzGameFramework
- AzQtComponents
- AzToolsFramework
- GridMate
- GridMateForTools

To declare a dependency on one of these Lumberyard modules, use the Waf `use` mechanism. The following example `wscript` file specifies a dependency on AzFramework.

```python
def build(bld):
    bld.DefineGem(
        # Add custom build options here
        includes = [bld.Path('Code/CryEngine/CryAction'),
                    use = ['AzFramework'],
    )
```

#### Adding a Dependency on Another Gem

You can configure gem A to depend on gem B by modifying gem A's `gem.json` and `wscript` files. The following example shows how the Twitch gem, included with Lumberyard, declares a dependency on the HttpRequestor gem (p. 1144). This `gems.json` file is located in the `lumberyard_installation\dev\Gems\Twitch` directory.

```json
{
    "GemFormatVersion": 3,
    "Uuid": "b63e64141fab40b791211ba257632e84",
    "Name": "Twitch",
    "DisplayName": "Twitch",
    "Version": "1.0.0",
    "LinkType": "Dynamic",
    "Summary": "Provides access to the Twitch Commerce SDK, social functions, login, chat, and other APIs.",
    "Tags": ["Twitch","Commerce","SDK","Social" ],
    "IconPath": "preview.png",
    "Dependencies": [
        {
            "Uuid": "28479e255bde466e91fc34ee808d9c7",
            "VersionConstraints": [ "->1.0" ],
            "_comment": "HttpRequestor"
        }
    ]
}
```
The Twitch gem specifies the HttpRequestor gem in the Dependencies section. The Dependencies section also has a VersionConstraints section that you can use to specify versioning requirements.

The Twitch gem's wscript file uses the use keyword to declare a dependency on HttpRequestor:

```python
def build(bld):
    import lumberyard_sdks
    file_list = []
    if lumberyard_sdks.does_platform_support_aws_native_sdk(bld):
        file_list.append('twitch.waf_files')
    else:
        file_list.append('lmbraws_unsupported.waf_files')

    bld.DefineGem(
        use = ['HttpRequestor', 'LmbrAWS'],
        uselib = [ 'TWITCHFSDK', 'AWS_CPP_SDK_CORE' ],
        includes = [],
        file_list = file_list,
        win_file_list = ['twitch_win.waf_files']
    )
```

Adding a Dependency on a Third-Party Library for Lumberyard

You can define a gem that uses a third-party library for Lumberyard that another module is also using. To do so, use the Waf uselib mechanism to add the dependency to the gem, as in the following example wscript file.

```python
def build(bld):
    bld.DefineGem(
        # Add custom build options here
        includes = [ bld.Path('Code/CryEngine/CryAction') ],
        use = [ 'AzFramework' ],
        uselib = [ 'AWS_CPP_SDK' ]
    )
```

For more information, see Adding Third-Party Libraries (p. 105) and Creating Third-Party Library Configuration Files for Waf (p. 108).

Note
Because the recursive nature of use can lead to linker errors, especially duplicate symbol errors, uselib is a better choice.

Using AZ Modules to Initialize Gems

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

AZ modules are code libraries designed to plug into Lumberyard games and tools. An AZ module is a collection of C++ code built as a static or dynamic library (.lib or .dll file) that implements specific initialization functions. When a Lumberyard application starts, it loads each module and calls these initialization functions. These initialization functions allow the module to connect to core technologies such as reflection, serialization, event buses (p. 1851), and the Working with component entities (p. 462).
Lumberyard uses the AZ module interface to extract the contents of your gem into the global environment. Each application has a single entity associated with it that is referred to as the system entity (p. 1104). AZ modules can add components to this entity before it is activated. Components that are added to the system entity are called system components. System components are often singleton/manager-type objects that aggregate or provide resources to game components or other systems. Like other entities, a system entity must have its dependencies present. You can assume that any systems that you depend on are booted and available when your system component is activated.

Modules are not a new concept in Lumberyard. In fact, the Lumberyard game engine is a collection of older style modules. These legacy modules have served the game engine well, but they have a number of shortcomings which are addressed by AZ modules, as presented in the next section.

Lumberyard currently supports both legacy modules and AZ modules but going forward will use AZ modules. Beginning in Lumberyard 1.5, a gem can contain AZ module code. Creating a new gem is the easiest way to get up a new AZ module up and running.

**Note**
AZ is the namespace of the AZCore C++ library upon which AZ modules are built. The letters AZ refer to Amazon; the term is a preview name that has nothing to do with Amazon Availability Zones and may be subject to change.

### Comparing AZ Modules to Legacy Modules

AZ modules have significant advantages over legacy modules, as the following table shows:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Legacy Modules</th>
<th>AZ Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatibility</td>
<td>Modules can be converted to AZ modules with no loss of functionality.</td>
<td>Anything that can be done in a legacy module can also be done in an AZ module. Most AZ module code could live within a legacy module, but legacy modules are not likely to be compatible with future AZ module–based Lumberyard tools.</td>
</tr>
<tr>
<td>Ease of adding services</td>
<td>Adding services usually requires editing files in CryCommon. A file for the singleton's class interface must exist in the CryCommon directory, and a variable to hold the singleton in gEnv must exist.</td>
<td>Modules create components and attach them to the system entity. No editing of game engine files is required.</td>
</tr>
<tr>
<td>(singleton classes) to modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of use for low-level</td>
<td>Modules load late, which prevents them from contributing low-level features to an application. All critical features must be in a single module that loads before others.</td>
<td>Modules load early in the application's startup sequence and are initialized in discrete stages. This allows <em>any</em> module to provide a low-level feature at an early stage that other modules can take advantage of later.</td>
</tr>
<tr>
<td>application features</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exposure of properties</td>
<td>Modules have no uniform way to let users control settings for their service. Some services read settings from .xml files in the</td>
<td>AZ modules expose the properties of system components to the Lumberyard reflection system. The reflection system makes information about</td>
</tr>
<tr>
<td>Game engine dependency</td>
<td>Modules must run in the game engine and are difficult to extend for use in tools that do not have game code.</td>
<td>Modules are not specific to the game engine and can be used outside it.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>Initialization functions</td>
<td>Function parameters are specific to CryEngine.</td>
<td>Function parameters are specific to the AZ framework; for more information, see the following section.</td>
</tr>
<tr>
<td>Order of initialization</td>
<td>Singleton code often depends on services offered by other singletons, so modules must be initialized in a very particular order. However, the order is not obvious. If someone is unfamiliar with the code in the modules, their loading order is difficult to ascertain.</td>
<td>Each module explicitly states its dependencies on system components. After all system components are examined, they are sorted according to these dependencies and initialized in the appropriate order. Each module is a first-class citizen.</td>
</tr>
</tbody>
</table>

### A Self-Aware Method of Initialization

Legacy modules are loaded in a particular order. Because **CrySystem** is loaded and initialized before the game module, it must provide all low-level systems such as logging and file I/O that a subsequent module might depend on. The game module itself cannot provide such low-level systems because it's initialized too late.

**AZ modules**, on the other hand, are all loaded as early as possible, and then initialized in stages. Because each module explicitly states its dependencies on system components, all system components can be examined beforehand, sorted according to dependencies, and initialized in the appropriate order (p. 1098). This makes it possible for low-level functionality (like a custom logging system) to be implemented from a game module. For more information about the initialization order of components, see **The AZ Bootstrapping Process** (p. 1112).

### Relationship with the AZ Framework

**AZ modules** are designed to work with the AZ framework, which is a collection of Lumberyard technologies such as reflection, serialization, **event buses** (p. 1851), and the **Working with component entities** (p. 462). The AZ framework supports game development but can also be used outside it. For example, Lumberyard tools like the Lumberyard Setup Assistant, **Using Asset Processor** (p. 248) and the component entity system use the AZ framework and AZ modules, but contain no game code. When the Resource Compiler builds slices, it loads AZ modules to extract reflection information about components within them.

AZ modules are code libraries that are built to use the AZ framework. When an AZ framework application loads an AZ module, the AZ module knows how to perform tasks such as gathering reflection information about the data types defined within that library.

### Smarter Singletons

AZ modules build their services (which are singleton classes) by using the same component entity system that Lumberyard uses to build in-game entities. A module simply places a system component on the system entity. This solves many of the problems associated with singletons in legacy modules.
The GUI in Lumberyard Editor uses the reflection system to expose the properties of entities (gameplay components) to designers. In the same way, Lumberyard uses the reflection system to expose the properties of system components so that you can customize your settings for a particular game. Because system components are really no different from gameplay components, you can use the Project Configurator to edit the properties of system components (p. 1104) just as you edit the properties of in-game components.

**Current Lumberyard AZ Modules**

The gems (p. 1100) provided with Lumberyard are all built as AZ modules. In addition, there are two AZ modules that are not built as Gems.

- **LmbrCentral**
  
  LmbrCentral contains components that wrap functionality from legacy modules. For example, the MeshComponent utilizes IRenderNode under the hood. LmbrCentral is used by game applications.

- **LmbrCentralEditor**
  
  Components can have editor-specific implementations that integrate with technology not available in the game runtime environment. Therefore, a separate module, LmbrCentralEditor, is used by Lumberyard Editor. This module contains all the code from LmbrCentral, plus code that is only for use in tools. The LmbrCentralEditor module is not for use in standalone game applications.

**Parts of an AZ Module, Explained**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

An AZ module has three key components: a class that inherits from AZ::Module, one or more public facing event buses, and a system component class.

This page describes module initialization, the use of system components as singletons, how EBus calls communicate with this singleton, and how to call the module externally after you have created it.

**The Module Class**

Each AZ module must contain a class that inherits from AZ::Module. When the module is loaded by an application, an instance of the class is created very early in the application's lifetime and its virtual functions are called at the appropriate times as the application goes through its bootstrapping process (p. 1112). This class reflects (p. 977) the components declared in the module and adds critical components to the system entity (p. 1104).

*Note*

At its core, every Lumberyard application has a single system entity. When a Lumberyard application starts, it creates the system entity. This entity's components, known as system components, power major systems within Lumberyard. The system entity always has the ID AZ::SystemEntityId (0).

The following skeleton code shows the basic structure of an AZ::Module class.

```cpp
namespace AZ {

/**
 * AZ::Module enables static and dynamic modules (aka LIBs and DLLs) to
 * connect with the running \ref AZ::ComponentApplication.
 */

```
* Each module should contain a class which inherits from AZ::Module.
* This class must perform tasks such as reflecting the classes within
  * the module and adding critical components to the system entity.

```cpp
class Module
{
public:
  Module();
  virtual ~Module();

  // Override to require specific components on the system entity.
  virtual ComponentTypeList GetRequiredSystemComponents() const;
};
```

The AZ::Module class exposes all points of integration with the AZ framework as virtual functions. These points of integration have been created as virtual functions on a class so that, whether initialization code is in a static or dynamic library, it's written the same way as much as possible. The very first actual initialization calls do need to be different for static and dynamic libraries. Lumberyard provides a macro to define this uninteresting glue code and let you write the interesting initialization code within your AZ::Module class.

We recommend that your AZ::Module class contain as little implementation code as possible. When the AZ::Module class is created, the application is just starting up and many systems are unavailable. If the AZ::Module class spawns a singleton or manager class, there is no guarantee that the systems on which this singleton relies will be ready for use. Instead, you should build your singletons as Lumberyard system components (p. 1098), which can control their initialization order.

Beginning in Lumberyard 1.5, gems are built using AZ modules. The following example "HelloWorld" AZ module was made by creating a new gem (p. 1064). The CryHooksModule class in this example is a helper wrapper around AZ::Module and provides your entire module access to gEnv.

```cpp
// dev/Gems/HelloWorld/Code/Source/HelloWorldModule.cpp
#include "StdAfx.h"
#include <platform_impl.h>
#include "HelloWorldSystemComponent.h"
#include <IGem.h>

namespace HelloWorld
{
  class HelloWorldModule : public CryHooksModule
  {
    public:
      AZ_RTTI(HelloWorldModule, "{39C21561-D456-413F-8C83-4214F6DBCA5}",
        CryHooksModule);
  
      HelloWorldModule()
      : CryHooksModule()
      {
        // Create descriptors for components declared within this module.
        m_descriptors.insert(m_descriptors.end(), {
          HelloWorldSystemComponent::CreateDescriptor(),
        });
      }

      // Add required system components to the system entity.
      AZ::ComponentTypeList GetRequiredSystemComponents() const override
      {
        return AZ::ComponentTypeList{
          azrtti_typeid<HelloWorldSystemComponent>(),
        };
      }
  }

  // dev/Gems/HelloWorld/Code/Source/HelloWorldSystemComponent.h
  // dev/Gems/HelloWorld/Code/Source/HelloWorldSystemComponent.cpp
```

Version 1.28
1094
The EBus

External code can call into your module, and receive events from your module, through the module's public event buses (p. 1851) (EBus). The EBus allows simple and safe function calls between different modules of code.

A new gem comes with one EBus by default, as shown in the following example.

```plaintext
// dev/Gems/HelloWorld/Code/Include/HelloWorld/HelloWorldBus.h
#pragma once
#include <AzCore/EBus/EBus.h>
namespace HelloWorld
{
    class HelloWorldRequests : public AZ::EBusTraits
    {
    public:
    #define EBUS_TRAITS_OVERRIDE_HEADER
        // EBusTraits overrides
        // These settings are for a "singleton" pattern.
        // A single handler can connect to the EBus.
        static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;
        // A single address exists on the EBus.
        static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
    #undef EBUS_TRAITS_OVERRIDE_HEADER

    // Put your public methods here
    virtual void SayHello(const char* name) = 0;
    
    using HelloWorldRequestBus = AZ::EBus<HelloWorldRequests>;
} // namespace HelloWorld
```

Calls to this EBus are handled by the system component, as described in the following section.

The System Component Class

Any major systems in your module that require a singleton should be built as system components. New gems come with a system component by default. The system component class is created during application startup and attached to the system entity (see GetRequiredSystemComponents() in HelloWorldModule.cpp).

In the current example, the system component class handles calls to the public EBus declared in HelloWorldBus.h. The following code shows the HelloWorldSystemComponent class.

```plaintext
// dev/Gems/HelloWorld/Code/Source/HelloWorldSystemComponent.h
#pragma once
#include <AzCore/Component/Component.h>
#include <HelloWorld/HelloWorldBus.h>
namespace HelloWorld
```
class HelloWorldSystemComponent : public AZ::Component, protected HelloWorldRequestBus::Handler
{
  public:
  // Every component definition must contain the AZ_COMPONENT macro,
  // specifying the type name and a unique UUID.
  AZ_COMPONENT(HelloWorldSystemComponent, "{72DFB0EE-7422-4CEB-9A40-426F26530A92}");

  static void Reflect(AZ::ReflectContext* context);
  static void GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided);
  static void GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible);
  static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required);
  static void GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent);

  protected:
  // AZ::Component interface implementation
  void Init() override;
  void Activate() override;
  void Deactivate() override;

  // HelloWorldRequestBus interface implementation
  void SayHello(const char* name) override;

};

namespace HelloWorld
{

  void HelloWorldSystemComponent::Reflect(AZ::ReflectContext* context)
  {
    // Reflect properties that game developers may want to customize.
    if (AZ::SerializeContext* serialize = azrtti_cast<AZ::SerializeContext*>(context))
    {
      serialize->Class<HelloWorldSystemComponent, AZ::Component>()
        ->Version(0)
        ->SerializerForEmptyClass();

      if (AZ::EditContext* ec = serialize->GetEditContext())
      {
        ec->Class<HelloWorldSystemComponent>("HelloWorld", "Says hello")
          ->ClassElement(AZ::Edit::ClassElements::EditorData, "")
          ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                      AZ_CRC("System"))
            ->Attribute(AZ::Edit::Attributes::AutoExpand, true);

      }
    }
  }

}
void HelloWorldSystemComponent::GetProvidedServices(AZ::ComponentDescriptor::DependencyArrayType& provided)
{
    provided.push_back(AZ_CRC("HelloWorldService"));
}

void HelloWorldSystemComponent::GetIncompatibleServices(AZ::ComponentDescriptor::DependencyArrayType& incompatible)
{
    // Enforce singleton behavior by forbidding further components
    // which provide this same service from being added to an entity.
    incompatible.push_back(AZ_CRC("HelloWorldService"));
}

void HelloWorldSystemComponent::GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required)
{
    // This component does not depend upon any other services.
    (void)required;
}

void HelloWorldSystemComponent::GetDependentServices(AZ::ComponentDescriptor::DependencyArrayType& dependent)
{
    // This component does not depend upon any other services.
    (void)dependent;
}

void HelloWorldSystemComponent::Init()
{
}

void HelloWorldSystemComponent::Activate()
{
    // Activate() is where the component "turns on".
    // Begin handling calls to HelloWorldRequestBus
    HelloWorldRequestBus::Handler::BusConnect();
}

void HelloWorldSystemComponent::Deactivate()
{
    // Deactivate() is where the component "turns off".
    // Stop handling calls to HelloWorldRequestBus
    HelloWorldRequestBus::Handler::BusDisconnect();
}

void HelloWorldSystemComponent::SayHello(const char* name)
{
    AZ_Printf("HelloWorld", "Hello %s, you certainly look smashing tonight.", name);
}

For more information about system components, see System Components (p. 1098).

**Calling the Module from External Code**

To call your module, invoke your public function through EBus. This example uses the SayHello function.

```cpp
#include <HelloWorld/HelloWorldBus.h>
```
void InSomeFunctionSomewhere()
{
    // ...
    // Invoke the call through EBus.
    EBUS_EVENT(HelloWorld::HelloWorldRequestBus, SayHello, "Bruce");
    // ...
}

System Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A traditional game engine contains many singleton classes, each in charge of a major system. In Lumberyard, these singletons are built using the same component entity system (p. 462) that powers gameplay entities. When an application starts, a system entity is created. Components added to this entity are known as system components. The system entity always has the ID AZ::SystemEntityId(0).

When you build singletons as Lumberyard system components, you are using a powerful suite of complementary technologies that facilitate problem resolution through established patterns. This topic describes system components in detail.

Smart Initialization Order

As a game engine grows in size, it tends to develop many singleton classes. A singleton class often requires communication with other singletons to function. This means that the order in which singletons are initialized is very important. Lumberyard solves this by building singletons as components.

A component can declare the services that it provides and the services on which it depends. When components are activated, they are sorted according to these declared dependencies, ensuring proper initialization order.

The following example shows two components that Lumberyard has ordered for initialization.

class AssetDatabaseComponent : public Component
{
    ...
    static void GetProvidedServices(ComponentDescriptor::DependencyArrayType& provided)
    {
        provided.push_back(AZ_CRC("AssetDatabaseService"));
    }
    ...
};
class AssetCatalogComponent : public AZ::Component
{
    ...
    static void GetRequiredServices(AZ::ComponentDescriptor::DependencyArrayType& required)
    {
        required.push_back(AZ_CRC("AssetDatabaseService"));
    }
    ...
};

The example shows the following:
- AssetDatabaseComponent is activated before AssetCatalogComponent.
- In the AssetDatabaseComponent class, the GetProvidedServices function reveals that the class provides a service called AssetDatabaseService.
- In the AssetCatalogComponent class, the GetRequiredServices function reveals that AssetCatalogComponent depends on AssetDatabaseService. Lumberyard understands this dependency and orders the initialization accordingly.

For more information about the initialization order of components, see The AZ Bootstrapping Process (p. 1112).

Easily Configurable Components

Often, a singleton has settings that are configurable for each game. It can be difficult for a low-level singleton to access configuration data if the system used to process this data hasn't started. Therefore, low-level singletons often rely on simple data sources such as command line parsers or .ini files.

A system component can expose its configuration through AZ reflection (p. 977). The Advanced Settings dialog box in the Project Configurator (p. 1104) uses this feature to enable the configuration of system components on a per-game basis. The Project Configurator saves an application descriptor file (p. 1111) that contains the settings for each system component, and this file is used to bootstrap the application and configure each component before it is activated. This is the same technology that the Entity Inspector (p. 475) uses to configure gameplay entities in Lumberyard Editor.

For more information, see Configuring System Entities (p. 1104).

Writing System Components

To designate a component as a system component, rather than a gameplay component, you must set the AppearsInAddComponentMenu field to System when you reflect to the EditContext.

The following example code designates the MemoryComponent as a system component.

```cpp
void MemoryComponent::Reflect(ReflectContext* context)
{
    if (SerializeContext* serializeContext = azrtti_cast<SerializeContext*>(context))
    {
        ... 
        if (EditContext* editContext = serializeContext->GetEditContext())
        {
            editContext->Class<MemoryComponent>("Memory System", "Manages memory allocators")
                ->ClassElement(AZ::Edit::ClassElements::EditorData,"")
                ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                AZ_CRC("System"))
                ... 
        }
    }
}
```

For more information on writing system components, see Creating System Components (p. 987).

Required System Components

Often, a module requires the existence of a system component. This requirement can be established through the module's GetRequiredSystemComponents() function. Any component type declared here is guaranteed to exist when the application starts.

In the following example, the Oculus gem requires the OculusDevice component.
AZ::ComponentTypeList OculusGem::GetRequiredSystemComponents() const override
{
    return AZ::ComponentTypeList{
        azrtti_typeid<OculusDevice>(),
    };
}

If a system component is optional, you can add it from Advanced Settings in the Project Configurator (p. 1104).

Gems and AZ Modules

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The gems system was developed to make it easy to share code between projects. Gems are reusable packages of module code and assets which can be easily added to or removed from a Lumberyard game. Gems also promote writing code in a way that is more modular than that found in legacy libraries. For example, each gem has its own include folder for its public interface code files. Gems also come with package management metadata such as semantic versioning and the ability to state dependencies on other gems.

Structure of a Gem

A gem's directory contents are organized as follows:

GemDirectory/
    Assets/ (assets usable to projects)
    Code/
        Include/ (public interface code files)
        Source/ (private implementation code files)
        Tests/ (code files for tests)
    wscript (waf build info)
    gem.json (gem metadata)

Waf Integration

Each game project must explicitly list the gems that it uses. When Using the Waf Build System (p. 63) runs, it builds only those gems which are actively in use. Waf also makes a gem's include/ directory accessible to any gems or projects that explicitly depend upon the gem.

Gems Built as AZ Modules

All gems that ship with Lumberyard are built as AZ modules. When you build a gem as an AZ module, the gem uses the initialization functions expected by the AZ framework. An AZ module gem has public interfaces that are event buses (p. 1851) and is better integrated with the new component entity system (p. 462).

When you use the Project Configurator to enable or disable a gem, Lumberyard updates the application descriptor file (p. 1111) accordingly to ensure it references all AZ modules. If you edit the dev \<project_asset_directory>\gems.json list of gems by hand, you can use the following command to bring the application descriptor file up to date:
About Gem Versioning

The GemFormatVersion value is versioning for how a gem is built. Gem version numbers like 0.1.0 refer to the gem's API version.

Gems from Lumberyard 1.4 and earlier (legacy gems) all have a GemFormatVersion value of 2. Starting in Lumberyard 1.5, all the gems included with Lumberyard are AZ modules and have a GemFormatVersion value of 3. This tells Lumberyard that the gem is an AZ module and that it should be loaded accordingly.

A gem may also have an API version number like 0.1.0. This is independent of the GemFormatVersion. The API version alerts your users to API changes. If the API version number changes, then users of the gem may need to make changes to their code. For example, the Rain gem will stay at version 0.1.0 until its API changes. If you were using the Rain gem from Lumberyard 1.4, you can still use the Rain gem from Lumberyard 1.5 without changing any of your data or code.

For more information about gems, see Add modular features and assets with Gems (p. 1064).

Creating an AZ Module That Is Not a Gem

AZ modules are in preview release and subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Beginning with Lumberyard 1.5, gems are AZ modules, so the preferred way to build an AZ module is to simply create a new gem. However, if your project requires an AZ module that must not be built as a gem, follow the steps provided here.

A. Start with a Gem

Because gems have all the required code for an AZ module, it's easier to create a gem first and then modify it not to be a gem. As an added convenience, the new gem names the code for you in an intuitive way. For an explanation of the code that you get in a new gem, see Parts of an AZ Module, Explained (p. 1093).

To create and modify a gem

1. First, create a gem by performing the following steps:
   a. Open the Lumberyard Project Configurator, located at \lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe. For example, when using Visual Studio 2017 as your build platform, the Project Configurator is located at \lumberyard-version\dev\Bin64\vc141\ProjectConfigurator.exe.
   b. Select your project (the default is SamplesProject).
   c. Click Enable Gems.
   d. Click Create a New Gem.
   e. Enter the name for your new module. (The example on this page uses the name “HelloWorld”.)
   f. Click Ok.
2. Move and rename the code directory from the new gem to your desired location. For example, move the directory...
dev/Gems/HelloWorld/Code

to

dev/Code/<optional subfolder>/HelloWorld

3. To remove the remaining noncode pieces of the gem, delete the directory dev/Gems/HelloWorld.

B. Modify the AZ Module Declaration

AZ modules that are not gems must not have UUIDs in their names, so you must modify the gem's .cpp file accordingly.

To modify the .cpp file

1. Remove the code that looks like the following:

   // DO NOT MODIFY THIS LINE UNLESS YOU RENAME THE GEM
   // The first parameter should be GemName_GemIdLower
   // The second should be the fully qualified name of the class above
   AZ_DECLARE_MODULE_CLASS(HelloWorld_010c14ae7f0f4eb1939405d439a9481a,
                           HelloWorld::HelloWorldModule)

2. Replace the AZ_DECLARE_MODULE_CLASS declaration with one that follows this syntax:

   AZ_DECLARE_MODULE_CLASS(HelloWorld, HelloWorld::HelloWorldModule)

   The first argument (HelloWorld) is a unique identifier to be included in your project.json file, and should match the target field of your wscript. You will do these steps later. The second argument is the same fully qualified name of the class already defined in your .cpp file.

C. Remove CryEngine References (Optional)

If your module does not access code from CryEngine (for example, it does not access gEnv), perform these additional steps.

To remove CryEngine references

1. Make the following changes to your .cpp file (in this example, HelloWorldModule.cpp).

   a. Remove #include <platform_impl.h>
   b. Remove #include <IGem.h>
   c. Add #include <AzCore/Module/Module.h>
   d. Change HelloWorldModule to inherit directly from AZ::Module instead of from CryHooksModule.

2. Remove the following include statement from the StdAfx.h file:

   #include <platform.h> // Many CryCommon files require that this be included first.

D. Modify the Wscript and Waf Spec Files

Next, you must modify the default wscript file to remove gem-specific commands, add your module directory to the wscript file, and add your module to the appropriate waf spec files (p. 74).
To modify the wscript and waf spec files

1. Modify the wscript contents to resemble the following:

```python
def build(bld):
    bld.CryEngineModule(
        target          = 'HelloWorld',
        vs_filter       = 'Game', # visual studio filter path
        file_list       = 'HelloWorld.waf_files',
        platforms       = ["all"],
        configurations  = ["all"],
        pch             = ["source/StdAfx.h"],
        use             = ["AzFramework"],
        includes        = ["include", 'source'],
    )
```

2. Modify the wscript in a parent directory so that waf recurses your module's directory, as in the following example.

```python
# ...
SUBFOLDERS = [
    # ...,
    'HelloWorld',
]
# ...
```

3. To enable waf to build your module, add the module to the appropriate waf spec files in your Lumberyard directory (`dev\WAF\specs\*.json`), as in the following example:

```json
{
    // ...
    "modules": {
        // ...
        "HelloWorld"
    }
    // ...
}
```

E. Configure Your Project to Load the New Module

When your project launches, it loads the modules listed in the `dev/<project_assets>/Config/Game.xml` file (the `Editor.xml` file is used when Lumberyard Editor is launched). These files are automatically generated and should not be edited by hand.

To configure your project to load your AZ module

1. To ensure your non-gem module is included in these automatically generated lists, add the following lines to your `project.json` file (path location `dev/<project_asset_folder>/project.json`):

```json
{
    // ...
    "flavors": {
        "Game": {
            "modules": [
                "LmbrCentral",
                "HelloWorld"
            ]
        }
    }
    // ...
}
```
Using AZ Modules to Initialize Gems

Note
The flavors section may be missing from your project. If it is not present, Lumberyard assumes that the LmbrCentral module is used for Game, and that the LmbrCentralEditor module is used for Editor.

2. From the dev directory, run the following command from a command prompt.

```
Bin64\lmbr.exe projects populate-appdescriptors
```

This command modifies the Game.xml and Editor.xml files to list the HelloWorld module.

F. Add the Module's Public Interfaces to Your Project's Include Paths

Finally, to make your AZ module's public interfaces available to the rest of your project, you must inform them project of your module's include directory.

To make your AZ modules public interfaces available to your project

- In your project's wscript file, edit the includes line to point to your project's include directory, as in the following example.

```
... includes = [..., bld.Path('Code/Engine/HelloWorld/include')], ...
```

Configuring System Entities

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A single system entity lives at the heart of every Lumberyard application. The system entity's components, known as system components (p. 1098), power major systems within Lumberyard. You can use the Project Configurator's System Entity Editor to choose and configure the system components for your project. Editing the system entity for your project is like editing an entity in the Entity Inspector (p. 475).

To configure system entities

1. Compile a profile build of your project so that the Project Configurator can load your project's compiled code.
2. Open the Lumberyard Project Configurator, located at lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe. For example, when using Visual Studio
2017 as your build platform, the Project Configurator is located at `lumberyard-version\dev\Bin64vc141\ProjectConfigurator.exe`.

3. In the Project Configurator, select your project.
4. Click **Advanced Settings**.

The first time that you use the System Entity Editor, it performs the following tasks:

1. Loads the configuration for your system entity.
2. Identifies system components that are missing.
3. Adds the missing system components to the configuration.
4. Displays the list in the **Added required components** dialog box. The following image shows an example.
Note
Some system components are optional, and some are required. Both the Lumberyard engine and the gems that your project uses can require certain components.

5. In the Added required components dialog box, click OK.
6. The Project option shows the project that you selected. Use the drop-down menu to select a different project to edit.

7. For the Configuration option, choose Game if you want to make changes to the system entity for the Game (launcher), or Editor to modify the system entity for the Editor.
8. To add a component to the system entity, click **Add Component**.
9. To see more information about a component before you add it, pause your pointer over the name.
10. To add the component to the system entity, select the component, and then click **Save**. The component appears in the list of components for the system entity.

11. To delete, disable, or change a component’s position, right-click the component and choose the corresponding action.
12. To configure system memory options, choose the **Memory Settings** tab.
13. Click **Save** to save your changes to disk. The changes that you make on the **System Entity** and **Memory Settings** tabs are saved to a Game.xml or Editor.xml application descriptor file.

**Application Descriptor Files**

When you edit a system entity's configuration with the Project Configurator **Advanced Settings** dialog box, you are actually editing an application descriptor file.

Application descriptor files list all modules that a project uses. Each project requires a Game.xml and an Editor.xml application descriptor file in its asset directory:

```
lumberyard_version\dev\project_asset_directory\Config\Game.xml
lumberyard_version\dev\project_asset_directory\Config\Editor.xml
```

In the Project Configurator **Advanced Settings** dialog box, these files correspond to the **Game** and **Editor** options in the **Configuration** menu.

The following example shows the beginning of a Game.xml file. Both the Game.xml file and the Editor.xml file have the same structure.

```xml
<ObjectStream version="1">
    <Class name="ComponentApplication::Descriptor"
      type="{70277A3E-2AF5-4309-9BBF-6161AFBDE792}"
        >
      <Class name="bool" field="useExistingAllocator" value="false" type="{A0CA880C-AFE4-43CB-926C-59AC48496112}"/>
```
The list of system components in the application descriptor file corresponds to the list of components on the System Entity tab in the Project Configurator, Advanced Settings, System Entity Editor dialog box. Each component can have its own settings. The application descriptor file also contains properties that determine how to allocate memory. These correspond to the settings on the Memory Settings tab in the System Entity Editor dialog box.

The AZ Bootstrapping Process

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

An AZ framework application initializes modules based on the dynamic libraries listed in the application descriptor file (p. 1111), and the static libraries referenced from the CreateStaticModules() function.

When an AzFramework::Application starts, the following order of events takes place:

1. The executable starts.
2. The AzFramework::Application class is initialized. It takes a path to an application descriptor file and a pointer to a function that will create the AZ::Modules from static libraries.
3. The application bootstraps itself just enough to read the application descriptor file.
4. The application descriptor file is read to get memory allocator settings and the list of dynamic libraries to load. Lumberyard is not yet able to read the system entity from the file.
5. Lumberyard shuts down the bootstrapped systems, configures them according to the settings it just loaded, and starts these systems back up.
6. Each dynamic library is loaded.
7. Each dynamic library's `InitializeDynamicModule()` function is run, which attaches the DLL to the global `AZ::Environment`.
8. Each static library's `AZ::Module` instance is created using the function pointer passed in during step 2.
9. Each dynamic library's `AZ::Module` instance is created by its `CreateModuleClass()` function.
10. Each AZ module's `RegisterComponentDescriptors()` function is called. Now the application knows how to serialize any components defined within a library.
11. The application descriptor file is read again to extract the system entity along with its components and their settings.
12. Each AZ module's `GetRequiredSystemComponents()` function is called. If any components are missing from the system entity, they are added.
13. The system entity is activated, and all of its system components are activated in the proper order.

At this point, initialization has been completed and the game is running.

## Gems Available in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard ships with the following Gems that are ready to be enabled:

**Topics**
- Asset Memory Analyzer Gem (p. 1115)
- Camera Framework Gem (p. 1121)
- ChatPlay Gem (p. 1121)
- Cloud Canvas Cloud Gems (p. 1122)
- Cloud Canvas Gem (p. 1122)
- CryEntity Removal Gem (p. 1123)
- EMotion FX Animation Gem (p. 1124)
- Fast Noise Gem (p. 1127)
- GameEffect Gem (p. 1129)
- GameLift Gem (p. 1130)
- GameState Gem (p. 1135)
- GameState Samples Gem (p. 1138)
- Gestures Gem (p. 1141)
- Graphics Scripting (p. 1143)
- HMD Framework Gem (p. 1143)
- HttpRequestor Gem (p. 1144)
• Image Processing (p. 1147)
• In-App Purchases Gem (p. 1147)
• Input Management Framework Gem (p. 1150)
• Landscape Canvas Gem (p. 1151)
• Lightning Arc Gem (p. 1152)
• Legacy Terrain Gem (p. 1159)
• LmbrCentral Gem (p. 1163)
• LyShine (p. 1163)
• Maestro Gem (p. 1163)
• Metastream Gem (p. 1164)
• Microphone Gem (p. 1171)
• Multiplayer Gem (p. 1171)
• Multiplayer Analytics Gem (p. 1179)
• NullVR Gem (p. 1180)
• NVIDIA Blast gem (p. 1181)
• NVIDIA Cloth gem (p. 1182)
• Physically Based Shaders (PBS) Gem (p. 1183)
• PhysX (p. 1188)
• PhysX Characters (p. 1188)
• PhysX Debug (p. 1188)
• Physics Entities Gem (p. 1189)
• Primitive Assets Gem (p. 1190)
• Process Life Management Gem (p. 1191)
• Python Asset Builder gem (p. 1192)
• RAD Telemetry Gem (p. 1192)
• Rain Gem (p. 1194)
• Render to Texture (p. 1197)
• SaveData Gem (p. 1198)
• Scene Processing Gem (p. 1201)
• Script Canvas Gem (p. 1201)
• Scripted Entity Tweener Gem (p. 1202)
• Sky Clouds Gem (p. 1202)
• Slice Favorites (p. 1203)
• Snow Gem (p. 1203)
• Starting Point Input Gem (p. 1206)
• Twitch v5 Gem (p. 1207)
• Twitch API Gem (p. 1207)
• Twitch Chat API Gem (p. 1208)
• UiBasics Gem (p. 1209)
• User Login Default Gem (p. 1209)
• Video Playback Gem (p. 1209)
• Video Playback Bink Gem (p. 1209)
• Virtual Gamepad Gem (p. 1210)
• Virtual Reality Gems (p. 1215)
• Visibility (p. 1215)
Asset Memory Analyzer Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Resource management is critical, particularly on platforms such as mobile devices and consoles where heap (system memory) and VRAM (video memory) are limited. In any given project, the model, texture, animation, and audio resource files that make up the project’s assets use the bulk of the memory allocated to run the project. **Asset Memory Analyzer** shows how memory is allocated to assets as your project runs. It is an indispensable tool to balance memory usage and get the best performance for your project.

The **Asset Memory Analyzer** is an Amazon Lumberyard Gem that displays a table of heap and VRAM memory allocations per asset through the **ImGUI** (Immediate Mode Graphical User Interface) overlay. In addition to live display of memory allocations for assets loaded in the project, the **Asset Memory Analyzer** can export allocation data to JSON and CSV files.

- Enable the **Asset Memory Analyzer** (p. 1115)
- View Live Asset Memory Analysis with **ImGUI** (p. 1116)
- Export an Asset Memory Analysis Snapshot to a File (p. 1117)
- View a JSON Asset Memory Analysis Snapshot in a Browser (p. 1118)
- Instrumenting Code for Asset Memory Analysis (p. 1119)

**Enable the Asset Memory Analyzer**

To use the **Asset Memory Analyzer**, enable asset scope tracking, configure and compile a profile build of your project, and enable asset memory analysis.

1. Use Project Configurator to add the **Asset Memory Analyzer** Gem and ImGUI Gem to your project.
2. Uncomment the line `enable_memory_tracking` and set its value to `True` in the `user_settings.options` file located in `\dev\WAF\` to enable asset scope tracking.

```plaintext
... ;use_crcfix = True
enable_memory_tracking = True
;generate_sig_debug_output = False
...
```

3. Configure your project.

   `lmbr_waf configure`

4. Create a **profile** build of your project.

   `lmbr_waf build_win_x64_vs2019_profile -p all --progress`

5. Set the CVAR `assetmem_enabled=1` via the Editor Console to enable asset memory analysis. To make the setting persistent, add `assetmem_enabled=1` to an appropriate config file such as `/dev/system_windows_pc.cfg`.

Version 1.28
1115
View Live Asset Memory Analysis with ImGUI

To view live asset memory allocation, enable the ImGUI overlay during gameplay and choose Asset Memory Analyzer in the ImGUI window.

1. In the Lumberyard Editor, press Ctrl+G or press the Play button to run your project.
2. Press the Home key to open the ImGUI overlay window.
3. Choose Asset Memory Analyzer from the top of the ImGUI overlay window.

Each recorded asset is displayed along with the number of allocations and total size in kilobytes for both heap and VRAM. Use the selections at the top of the ImGUI overlay window to sort the table by heap size, heap count, VRAM size, VRAM count or by asset label alphabetically.

**Note**

The asset label is the full path for each asset from the root of the project folder.

Click the arrow to the left of an asset to expand it and view individual allocations and references belonging to the asset.
Export an Asset Memory Analysis Snapshot to a File

Snapshots of asset memory allocation can be exported to JSON or CSV files through three methods:

- Click Asset Memory Analyzer in the ImGui overlay window and choose Export JSON or Export CSV.
- Use the console commands assetmem_export_json or assetmem_export_csv in the Editor Console to generate the file.
- Call ExportJSONFile or ExportCSVFile on the AssetMemoryAnalyzerRequestBus in C++, with nullptr as the parameter, to generate the file in the default location.

EBUS_EVENT(AssetMemoryAnalyzerRequestBus, ExportJSONFile, nullptr);
The snapshot file is created in the project log directory located at `dev/Cache/ProjectName/pc/user/log` and named `assetmem-2020-01-30-11-45-25.json` or `.csv`.

**Note**
Due to the limitations of the CSV format, only a top-level overview of assets will be written to CSV, *without* the hierarchical drill-down available in the JSON file or the ImGui overlay window.

**View a JSON Asset Memory Analysis Snapshot in a Browser**

JSON snapshots can be viewed in a browser with a web viewer provided with Lumberyard. The web viewer is located at `\dev\Gems\AssetMemoryAnalyzer\www\AssetMemoryViewer\index.html`. Open the `index.html` file and drag-and-drop the JSON file onto the page, or click on the target area to browse to it. This displays the contents of the file in an expandable table.

**Note**
Chromium based browsers are most reliable.

The table can be sorted by any column. The columns give a breakdown by multiple categories:

- **Heap** allocations and VRAM allocations
- **Local** summary not including any sub-assets and **Total** summary including all sub-assets
- **Number** of allocations and **Kilobytes** allocated

Expanding assets will display individual allocations belonging to the asset and sub-assets that were loaded through references.

The fields at the top of the table can be used to filter the assets by their label on a number of conditions, including regular expressions.
Instrumenting Code for Asset Memory Analysis

Initial Loading of an Asset

The Asset Memory Analyzer traps allocations (heap and VRAM) that occur during a slice of code execution or scope when an asset is active for recording. When a system begins loading a new asset, it should use the AZ_ASSET_NAMED_SCOPE macro to demarcate the C++ scope in which that asset may be actively making allocations. For example:

```cpp
#include <AzCore/Debug/AssetMemoryDriller.h>

Foo* LoadMyFooAsset(const char* name)
{
    AZ_ASSET_NAMED_SCOPE("Foo: %s", name);
    Foo* result = aznew Foo(name);  // The call to aznew will be recorded as associated with the asset "Foo: <name>"
    return result;  // Once we exit this function, the asset will no longer be in scope, and subsequent allocations will not be recorded
}
```

Subsequent Processing of an Asset

When the asset is being updated, otherwise processed, or is being handed off to a different thread, it should use the AZ_ASSET_ATTACH_TO_SCOPE macro with a pointer that was allocated and tracked by the initial asset. This will associate any further allocations with the same asset until the scope of the declaration closes.

```cpp
#include <AzCore/Debug/AssetMemoryDriller.h>

void UpdateAllFoos(const AZStd::vector<Foo*>& allFoos)
{
    for (Foo* foo : allFoos)
    {
        AZ_ASSET_ATTACH_TO_SCOPE(foo);  // Subsequent allocations in this scope will associate with any asset that was in scope when foo was allocated
        UpdateFoo(foo);
    }
}

void UpdateFoo(Foo* foo)
{
    aznew Bar;  // This automatically gets recorded with the owning asset for foo
    AZStd::thread doThreadedWork([foo]()
    {
        // Work being done on a different thread means we need to reattach to the owning asset
        AZ_ASSET_ATTACH_TO_SCOPE(foo);
        aznew Bar;  // This will now be recorded under the owning asset for foo
    });
    doThreadedWork.join();
}
```

You can attempt to attach to any pointer that was created while that asset was in scope, or even any portion of memory that was allocated to it.

For instance, the following code is valid:
```cpp
#include <AzCore/Debug/AssetMemoryDriller.h>

struct Baz
{
    int a;
    char* b;
    double c;
};

Baz* CreateBaz(const char* name)
{
    AZ_ASSET_NAMED_SCOPE(name);
    Baz* baz = aznew Baz;  // bar is associated with the named asset
    return baz;
}

void TestScopes()
{
    Baz* baz = CreateBaz("My test baz");
    
    {  
        AZ_ASSET_ATTACH_TO_SCOPE(&baz->c);  // This works, even though "c" didn't have its
        own allocation
        baz->b = aznew char[32];  // This allocation will be recorded under the asset "My
        test baz"
    }
}
```

An original pointer to an object that was allocated within a scope is not required in order to attach to it. This makes it possible to attach across systems to objects that have been defined with multiple inheritance.

### EBus Processing of and Asset

EBus handlers can automatically attempt to attach to a scope for each handler receiving an event. This works when the handler was allocated as part of an asset.

If the handler was created while an asset was in scope, you can modify an EBus as follows:

```cpp
#include <AzCore/Debug/AssetMemoryDriller.h>

class MyEvents : public AZ::EBusTraits
{
    // Process individual events by first attempting to attach to the asset that owns the handler
    template<typename Bus>
    using EventProcessingPolicy = Debug::AssetTrackingEventProcessingPolicy<Bus>;

    // Regular Ebus definitions
    virtual void MyFunction() = 0;
};
```

Some Lumberyard EBuses use this feature, such as the **TickBus**. If you find others that should use it, please add them! You should not default to using this `EventProcessingPolicy` if it is not applicable.

Instrumentation does create some overhead which can negatively affect your project's performance.

Creating a new named scope requires:
• function calls
• an environment lookup
• locking a mutex
• two hashtable lookups
• thread-local modifications

Attaching to an existing scope requires:

• function calls
• an environment lookup
• locking a mutex
• a lookup in a large red-black tree
• thread-local modifications

Most often this is a relatively small cost, but it is significant enough that you should not use the
AZ.Asset.Attach.To.Scope macro or use the AssetTrackingEventProcessingPolicy on your
EBus gratuitously, especially if it is unlikely to attach to anything.

When asset tracking is disabled, i.e. if the AZ.Analyze.Asset.Memory macro remains undefined, there
is zero cost to instrumentation of scopes. This is default in performance builds.

Note
EBuses that use the AssetTrackingEventProcessingPolicy will still have the indirection
of a function call for each handler on the bus in debug builds only. Non-debug builds inline this
function call away.

Camera Framework Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

The Camera and Camera Framework Gems are a base upon which you can build more complex camera
systems. This gem contains the Camera (p. 566) and the Camera Rig (p. 570), which work together to
define a basic camera and its control rig. You can customize the camera rig component through three
different behaviors:

• Target acquiring behavior
• Target transform modifying behaviors
• Final camera transform modifying behaviors

ChatPlay Gem

The ChatPlay Gem is deprecated and will be removed in a future version of Lumberyard. It has been
replaced by the Twitch Chat API Gem.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.
The Twitch ChatPlay Gem provides a flexible framework to create customized game interactions between broadcasters and spectators on Twitch, the world’s leading social video platform and community for gamers. Twitch ChatPlay includes support for chat commands, polls, and surveys that can be triggered by Twitch viewers through the Twitch chat channel.

For example, you can create a chat command #cheer that triggers celebration animations in your game.

**Additional Links**

- Intro to ChatPlay

**Cloud Canvas Cloud Gems**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A Cloud Canvas cloud gem includes everything required for you to include AWS cloud-connected functionality in your project. A cloud gem is a package of specific cloud-connected functionality, assets and AWS resource definitions.

**Cloud Gems Included with Lumberyard**

Lumberyard includes the following cloud gems. You can enable them in a project by using the Lumberyard Project Configurator (p. 43) tool. For more information, visit the following links.

- **Dynamic Content** (p. 2155) – Allows Lumberyard .pak files that contain new and updated game assets to be uploaded to the cloud for subsequent automatic download to the game client.
- **Player Account** (p. 2188) – Provides a standalone player authentication and management solution that uses Amazon Cognito.
- **Web Communicator** (p. 2204) – Send AWS cloud service events to your game's connected editors or clients. Your editors or game clients can use this information to update themselves without having to poll AWS for updates.

AWS services accessed through Cloud Canvas may be subject to separate charges and additional terms. For more information, see Pricing (p. 2115).

**Cloud Canvas Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Cloud Canvas Gem enables you to use Cloud Canvas visual scripting to AWS services. With Cloud Canvas you can build connected game features that use Amazon DynamoDB (DynamoDB), Amazon Lambda, Amazon Simple Storage Service (Amazon S3), Amazon Cognito, Amazon Simple Notification Service (Amazon SNS), and Amazon Simple Queue Service (Amazon SQS). You can also create cloud-hosted features such as daily gifts, in-game messages, leaderboards, notifications, server-side combat resolution, and asynchronous multiplayer gameplay (e.g. card games, word games, ghost racers, etc.). Cloud Canvas eliminates the need for you to acquire, configure, or operate host servers yourself, and reduces or eliminates the need to write server code for your connected gameplay features.
AWS services accessed via Cloud Canvas may be subject to separate charges and additional terms. For more information, see Implementing Connected Features with Cloud Canvas (p. 2106).

**CryEntity Removal Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the CryEntity Removal gem to disable all legacy features from Lumberyard Editor. Legacy features appear as legacy in Lumberyard Editor and will eventually be removed. This includes the following features:

- **Database View** (p. 1264) (Legacy)
- **Object Selector** (Legacy)
- **Layer Editor** (Legacy)
- **Rollup Bar** (Legacy)
- **Asset Browser** (Legacy)

When the gem is enabled, Lumberyard Editor displays only the features and tools that use the new component entity system. For more information, see Working with component entities (p. 462).

By default, the gem is disabled. For more information about enabling gems, see Add modular features and assets with Gems (p. 1064).

**Note**

- When you enable the gem and open Lumberyard Editor, you are prompted to convert your legacy entities (CryEntities) to the new component entity system. Once an entity is converted, it cannot be converted back to a legacy entity. For more information about converting your entities, see Converting Entities with the Legacy Converter (p. 963).
- If you want to enable the legacy features again, return to the Project Configurator and disable the CryEntity Removal gem. When you reload Lumberyard Editor, the legacy features reappear.
- If you create a project with Lumberyard 1.12 or newer, the CryEntity Removal gem is enabled by default. If you want to use legacy features, disable the gem.
EMotion FX Animation Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With the EMotion FX Animation gem, you can use the EMotion FX Animation Editor character animation system. The node-based system provides an intuitive approach to controlling character behavior from animation graphs. You can visually build hierarchal state machines and blend trees and use parameter controls for interactive testing. Character motion is controlled through motion extraction.

The character animation system includes tools for building the following:

- Transition conditions
- Blend spaces
- Sync tracking
- Motion events
- Mirroring animation
- Controllers

The Animation Editor is compatible with .fbx files and converts these files to the .actor and .motion formats.

For more information, see Create and animate characters (p. 1340).

Enable this gem for your project in the Project Configurator. For more information, see Creating a Game Project in Lumberyard (p. 45).
Lumberyard version 1.25 introduces deferred initialization for unique data in anim graphs. Rather than preallocating unique data for the whole anim graph upfront, allocation is deferred until the objects are used for the first time. For example, when transitioning into a given state, the unique data of the transition target is allocated and initialized right before the transition starts blending. This decreases instantiation time and makes spawning new characters that run an anim graph faster. The bigger the anim graph, the greater the speed improvement.

**Changes and using unique data in custom nodes**

- Unique data are no longer pre-allocated for the whole anim graph.
- Unique data are now allocated when they are needed and requested via `AnimGraphInstance::FindOrCreateUniqueData()`.
- Data shared across characters using the same anim graphs should be members of the anim graph node; for example, the name of the joint a look at node operates on.
- Instanced data belongs inside the node's unique data class. For example, the actual joint index based on the joint name stored in the node, as anim graphs can be shared across characters that have different skeletons.
• **UniqueData::Update()** is the place where the shared data is read and prepared to be used in an optimal way on the character instance. For example, this is where we would find the joint index for the given character instance, based on the shared joint name.

• **Update()** is called if the unique data is invalidated and requested for use with **AnimGraphInstance::FindOrCreateUniqueData()**.

• When a new joint name is selected in the UI, the unique datas for all currently present anim graph instances are invalidated and **Update()** is called to update all the joint indices.

• Newly created unique datas are invalidated by default and are updated automatically before their first usage.

### Accessing unique datas from within a custom node

There are two ways to access unique datas:

- **AnimGraphInstance::FindOrCreateUniqueData()**: Use this in your node's **Update()**, **TopDownUpdate()**, **PostUpdate()** or **Output()** functions. This guarantees that you have access to up-to-date unique data. If the unique data doesn't exist yet, it's created and initialized. If the unique data does exist, its pointer is available. **UniqueData::Update()** is called automatically if the unique data is invalidated when requesting it. The resulting object is valid and current in all ways.

- **AnimGraphInstance::GetUniqueObjectData()**: Use this for direct access to the unique data for the given node. If the unique data does not exist, it returns a nullptr. The unique data might also be invalidated and outofdate. **AnimGraphInstance::GetUniqueObjectData()** might be used in the **Rewind()** method since it's not required to create or update the unique data if the node hasn't been active yet.

### Port an existing custom node to version 1.25

1. Remove the mMustUpdate flag from the unique data if you use the flag to skip heavy operations when shared data changes (a picked joint in the LookAtNode, for example).

2. Remove the mIsValid flag from the unique data and use the error flag m_hasError from the base **AnimGraphObjectData**.

   **Note**
   If you need to populate the error flag up/down the hierarchy, don't check and set it inside **UniqueData::Update()**. Instead, call **AnimGraphNode::SetHasError()** inside the node to automatically populate the update flag.

3. If you use **AnimGraphNode::SetHasError()**, be aware the first parameter has changed from an anim graph instance to a unique data pointer for performance reasons.

4. Override **AnimGraphObject::CreateUniqueData()** for your custom node and return a newly allocated node-custom unique data. No exceptions, the method should always return a newly allocated unique data.

```cpp
AnimGraphObjectData* CreateUniqueData(AnimGraphInstance* animGraphInstance) override {
    return aznew UniqueData(this, animGraphInstance);
}
```

5. Remove the **YourCustomNode::UpdateUniqueData()** and move its contents to **UniqueData::Update()**.

   **Note**
   • There is no need to check for mMustUpdate anymore. We removed this flag in step 1.
• There is no need to manually call UniqueData::Update() manually inside any of the methods called at runtime: Update(), PostUpdate(), Output(). Whenever you retrieve the unique data with AnimGraphInstance::FindOrCreateUniqueData(), it is up to date.

• If you need access to the node itself, you can do this with the following code inside your unique data's Update() method:

```cpp
YourCustomNode* customNode = azdynamic_cast<YourCustomNode*>(mObject);
AZ_Assert(customNode, "Unique data linked to incorrect node type.");
```

6. Make sure to call YourCustomNode::Reinit when a UI element changes that should invalidate the unique datas. This can be done via:

```cpp
>Attribute(AZ::Edit::Attributes::ChangeNotify, &YourCustomNode::Reinit)
```

7. AnimGraphObject::Reinit() default implementation invalidates unique datas for all existing anim graph objects for the called object. Override Reinit() only if you need to do additional things and make sure to call Reinit() from the base class. Remove Reinit() from your custom node if all you need to do is invalidate the unique data of the node.

**Fast Noise Gem**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The FastNoise Gradient Gem uses the third-party, open source FastNoise library to provide a variety of high-performance noise generation algorithms in Amazon Lumberyard. You can use noise generation to generate content in a game.

The FastNoise Gradient Gem provides the FastNoise Gradient (p. 590) component which expresses noise generation algorithms as Gradient signals. Any system that's compatible with the Gradient Signal Gem, such as Vegetation, can use noise generation.

Gradient signals are values that range from 0.0 to 1.0 and are automatically mapped to world positions. They are typically visualized as either greyscale images or waveforms.

Examples of noise algorithms:
Enable the FastNoise Gradient Gem

To make the FastNoise Gradient component available in Lumberyard, you must build and configure your project with the FastNoise Gradient Gem enabled.

**To enable the FastNoise Gradient Gem**

1. Use Project Configurator to add the FastNoise Gradient Gem to your project.
2. Configure your project. Use the following command.

   ```
lmbr_waf configure
   ```

3. Build your project. Use the following command.

   ```
lmbr_waf build_win_x64_vs2019_profile -p all --progress
   ```

For more information on Gems, see the Gems documentation (p. 1064).

GameEffect Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Game Effect System Gem provides fundamentals for creating and managing the visual effects of the Lightning Arc Gem. If you install the Lightning Arc Gem (p. 1152), you must also install the Game Effect System Gem. The Lightning Arc Gem is the only Lumberyard gem that is dependent on the Game Effect System Gem.
GameLift Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Amazon GameLift is an AWS service for deploying, operating, and scaling session-based multiplayer games. With Amazon GameLift, you can quickly scale high-performance game servers up and down to meet player demand without any additional engineering effort or upfront costs.

The GameLift Gem provides a framework to extend Lumberyard networking (GridMate) to work with GameLift resources via GameLift server and client APIs.

Getting Started

To use the Amazon GameLift service with your Lumberyard project, you simply enable the GameLift Gem using Project Configurator.

You will want to familiarize yourself with GameLift concepts found in the Amazon GameLift Developer Guide. GameLift resources like fleet, queue, matchmaking rule, and config need to be deployed prior to running your game on GameLift. For more details, refer to the GameLift CLI.

Implementation

The GameLift Gem provides server and client code. Server code interacts with GameLift server SDK, while client code interacts with the GameLift client from the AWS native SDK.

The GameLift Gem server code is responsible for initializing the server process by initializing the GameLift server SDK and then hosting game sessions for the clients to connect. If run in custom matchmaking backfill configuration, the server is also responsible for starting and cancelling matchmaking backfill requests.

The GameLift Gem client provides options to create, search, join, and matchmaking into GameLift game sessions. Matchmaking is handled using GameLift’s FlexMatch matchmaking system. Clients can create, search, and join game sessions directly on GameLift fleets, or using preconfigured GameLift queues. If matchmaking, clients can use a preconfigured matchmaking configuration to create or join game sessions.

On the server side, the GameLift Gem provides the GameLiftServerService. On the client side, the Gem provides the GameLiftClientService. When you use the GameLift Gem in your game project, the following sequence of events occurs:

- The server starts the GameLiftServerService and listens for events delivered by the GameLift Server SDK.
- The client starts the GameLiftClientService, sends a GameLiftSessionRequest or GameLiftGameSessionPlacementRequest, and searches and joins game sessions using the fleet ID or queue name. The client service also provides an option to create or join game sessions using FlexMatch matchmaking.

These events are illustrated in the following workflow diagrams.

GameLift client-server communication workflow using fleet or queues to create a game session.
GameLift client-server communication workflow using FlexMatch, with automatic or custom backfill.
Sample Code

The GameLift code in this section follows the workflow illustrated in the preceding diagram and is separated into server-side code and client-side code. The code is enabled only when BUILD_GAMELIFT_SERVER and BUILD_GAMELIFT_CLIENT are defined.

Server-side Code

Use the following sample code as a guide when starting and hosting a GameLift server session.

Start GameLiftServerService

```
GridMate::GameLiftServerServiceDesc serviceDesc;
serviceDesc.m_port = settings.m_serverPort;
if (settings.m_logPath)
{
    serviceDesc.m_logPaths.push_back(settings.m_logPath);
}

m_service = GridMate::StartGridMateService<GridMate::GameLiftServerService>(m_gridMate,
    serviceDesc);
```

Host a Session

```
GridMate::CarrierDesc carrierDesc;
carrierDesc.m_port = s_gameLiftSettings.m_serverPort;
carrierDesc.m_driverIsFullPackets = false;
carrierDesc.m_driverIsCrossPlatform = true;

GridMate::GameLiftSessionParams sp;
sp.m_topology = GridMate::ST_CLIENT_SERVER;
sp.m_gameSession = &gameSession;

EBUS_EVENT_ID_RESULT(session, m_gridMate, GridMate::GameLiftServerServiceBus, HostSession,
    sp, carrierDesc);
```

Start/Stop Matchmaking Backfill

```
EBUS_EVENT_ID(m_gridMate, GridMate::GameLiftServerServiceBus, StartMatchmakingBackfill,
    m_updateGameSession, m_session, checkForAutoBackfill=true);

EBUS_EVENT_ID(gEnv->pNetwork->GetGridMate(), GridMate::GameLiftServerServiceBus,
    StopMatchmakingBackfill, m_session, m_ticketId);
```

Client-side Code

Use the following sample code as a guide when using the GameLift client service.

Start GameLiftClientService

```
GridMate::GameLiftClientServiceDesc serviceDesc;
serviceDesc.m_accessKey = settings.m_accessKey;
serviceDesc.m_secretKey = settings.m_secretKey;
serviceDesc.m_endpoint = settings.m_endpoint;
serviceDesc.m_region = settings.m_region;

m_service = GridMate::StartGridMateService<GridMate::GameLiftClientService>(m_gridMate,
    serviceDesc);
```
Send GameLiftSessionRequest / GameLiftGameSessionPlacementRequest

```cpp
GridMate::GameLiftSessionRequestParams reqParams;
reqParams.m_instanceName = "TestSession";
reqParams.m_numPublicSlots = 16;
reqParams.m_params[0].m_id = "param1";
reqParams.m_params[0].m_value = "value12";
reqParams.m_numParams = 1;

// Only need ONE of the below. Queue gets the highest preference, followed by alias and fleet id.
reqParams.m_fleetId = "fleet_id";
reqParams.m_aliasId = "alias_id";
reqParams.m_queueName = "queue_name";

m_sessionRequest = m_service->RequestSession(reqParams);
EBUS_EVENT_ID_RESULT(m_session, m_gridMate, GridMate::GameLiftClientServiceBus, RequestSession, reqParams);
```

Search Active Sessions

```cpp
GridMate::GameLiftSearchParams searchParams;
// Only need ONE of the below. Queue gets the highest preference, followed by alias and fleet id.
searchParams.m_fleetId = "fleet_id";
searchParams.m_aliasId = "alias_id";
searchParams.m_queueName = "queue_name";

EBUS_EVENT_ID_RESULT(m_search, m_gridMate, GridMate::GameLiftClientServiceBus, StartSearch, GridMate::GameLiftSearchParams());
```

Join a Session

```cpp
GridMate::CarrierDesc carrierDesc;
carrierDesc.m_port = 33435;
carrierDesc.m_enableDisconnectDetection = true;
carrierDesc.m_connectionTimeoutMS = 10000;
carrierDesc.m_threadUpdateTimeMS = 30;

const GridMate::GameLiftSearchInfo& gameLiftSearchInfo = static_cast<const GridMate::GameLiftSearchInfo&>(*gridSearch->GetResult(0));
EBUS_EVENT_ID_RESULT(m_session, m_gridMate, GridMate::GameLiftClientServiceBus, JoinSessionBySearchInfo, gameLiftSearchInfo, carrierDesc);
```

Join using FlexMatch Matchmaking

```cpp
m_matchmakingConfigName = "MatchmakingConfig";
EBUS_EVENT_ID_RESULT(m_search, m_gridMate, GridMate::GameLiftClientServiceBus, StartMatchmaking, m_matchmakingConfigName);
```

GameState Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The GameState gem helps you manage and determine, at a high level, the state that the game is in. Because the GameState gem uses a stack to manage game states, returning to a previous state is straightforward.

For information about sample game states and enabling the GameState Samples gem, see Enabling Gems (p. 1064).

**Note**
For a sample implementation of game states, see the GameState Samples Gem (p. 1138). The GameState Samples Gem depends on the GameState gem. You can customize the game states in the GameState Samples gem to meet the requirements of your game and communicate with your game code.

**Examining the Code**

The GameState gem manages a stack (or pushdown automaton) of abstract game states. The GameState gem includes the following code members:

- **IGameState** – Abstract interface that all concrete game state classes must be derived from.
- **GameStateRequests** – EBus (p. 1851) interface that other systems use to submit requests related to the game state.
- **GameStateNotifications** – EBus interface that other systems use to listen for events related to the game state.
- **GameStateSystemComponent** – Implements the GameStateRequestBus interface and sends events over the GameStateNotificationBus.

**IGameState**

IGameState is the abstract interface that all concrete game state classes must be derived from. The interface defines methods that track changes in game state, as seen in the following excerpt from the source code at

`lumberyard_version\dev\Gems\GameState\Code\Include\GameState\GameState.h`

```cpp
// Called when this game state is pushed onto the stack.
virtual void OnPushed() {}

// Called when this game state is popped from the stack
virtual void OnPopped() {}

// Called when this game state is set as the active game state.
virtual void OnEnter() {}

// Called when this game state is replaced as the active game state.
virtual void OnExit() {};
```
Lumberyard User Guide

GameState Gem

//! Called each frame while this game state is the active game state.
virtual void OnUpdate() {};

GameStateRequests

The methods in the GameStateRequests EBus perform essential tasks like creating, pushing, and popping a game state, or getting the active game state or active game state type. For the complete source code, see the [lumberyard_version]dev\Gems\GameState\Code\Include\GameState\GameStateRequestBus.h file.

//! Create a new game state.
//! \tparam GameStateType - The game state type to create.
//! \param[in] checkForOverrides - True to should check for an override, false otherwise.
//! \return - A shared pointer to the new game state that was created.
template<class GameStateType>
static AZStd::shared_ptr<IGameState> CreateNewOverridableGameStateOfType(bool checkForOverride = true);

//! Create a new game state and push it onto the stack to make it the active game state.
//! New game states are created and stored in the stack using a shared_ptr, so they are
//! destroyed automatically after they are popped off the stack (assuming that nothing
//! else retains a reference – for example, through GameStateNotifications::OnActiveGameStateChanged).
//! \tparam GameStateType - The game state type to create and activate.
//! \param[in] checkForOverrides - True to check for an override, false otherwise.
template<class GameStateType>
static void CreateAndPushNewOverridableGameStateOfType(bool checkForOverride = true);

//! Pop game states from the stack until the active game state is of the specified type.
//! \tparam GameStateType - The game state type in the stack that you want to be active.
//! \return True if the active game state is now of the specified type, false otherwise.
template<class GameStateType>
static bool PopActiveGameStateUntilOfType();

//! Query whether the active game state is of the specified type.
//! \tparam GameStateType - The game state type to check whether it is active.
//! \return - True if the active game state is of the specified type, false otherwise.
template<class GameStateType>
static bool IsActiveGameStateOfType();

//! Query whether the game state stack contains a game state of the specified type.
//! \tparam GameStateType - The game state type to check whether it is in the stack.
//! \return - True if the stack contains a game state of the specified type, false otherwise.
template<class GameStateType>
static bool DoesStackContainGameStateOfType();

//! Update the active game state. Called during the AZ::ComponentTickBus::TICK_GAME
//! priority update of the AZ::TickBus, but can be called independently any time if needed.
virtual void UpdateActiveGameState() = 0;

//! Request the active game state (if any)
//! \return - A shared pointer to the active game state (empty if there is none).
virtual AZStd::shared_ptr<IGameState> GetActiveGameState() = 0;

//! Push a game state onto the stack, making it become the active game state.
//! If newGameState is already found in the stack, the call fails and returns false.
//! However, it is possible for multiple instances of the same game state type to occupy the stack.
//! \param[in] newGameState - The new game state to push onto the stack.
//! \return - True if the game state was successfully pushed onto the stack, false otherwise.
virtual bool PushGameState(AZStd::shared_ptr<IGameState> newGameState) = 0;
GameState Samples Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The GameState Samples gem uses the GameState Gem (p. 1135) to provide a set of sample game states that control the high-level flow of a game.

For information about enabling the GameState Samples gem, see Enabling Gems (p. 1064).
Game States Included

The GameState Samples gem includes the following game states. These states commonly occur in the beginning, middle, and end of a game.

- **Main menu state** – Enables any level in the project to be loaded from a button click.
- **Level loading state** – Displays a placeholder loading screen.
- **Level running state** – Active while the game is running.
- **Level paused state** – Enables resuming or returning to the main menu to select another level.
- **Other states** – Game states that react to user sign-in and sign out and controller connections and disconnections.

Flow of Game States

The following diagram shows the flow of the default game states in the GameState Samples gem.
Possible Uses

The following are some possible ways to use the GameState Samples gem:

- **Copy** – Copy the gem to your game project to use as a starting point for further customization. This method offers the most flexibility. However, you should not enable the GameState Samples gem in Project Configurator. Instead, copy each class in the gem that inherits from `IGameState` into your game project. (For the source code, see the `lumberyard_version\dev\Gems\GameStateSamples`
The advantage of this approach over either of the following options is that you can freely modify the samples to fit the specific needs of your game.

- **Derive** – Derive from the code to create your own game states. This approach is recommended if you want to keep the same behavior as the sample game states, but with only minor customizations. For example, you could create a main menu class like the following:

  ```c++
  MyCustomMainMenu : public GameStateMainMenu
  ```

  You could then customize the class through inheritance to load a different main menu UI Canvas. The disadvantage of the "derivation" approach is that it places some logic in the gem and the rest in your game code. This can make your solution hard to follow or debug.

- **Modify** – Modify the GameState Samples gem directly. Because gems can't depend on the game, and therefore can't effectively communicate with any game-specific code, this option is not recommended.

### Gestures Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Gestures gem to recognize common gesture-based input, which includes the following:

- Tap or click – Single-touch, discrete gesture
- Drag or pan – Single-touch, continuous gesture
- Hold or press – Single-touch, continuous gesture
- Swipe – Single-touch, discrete gesture
- Pinch – Multiple-touch, continuous gesture
- Rotate – Multiple-touch, continuous gesture

Single-touch gestures (such as tap, drag, hold, and swipe) can be detected with touch or mouse input. However, multiple-touch gestures (such as pinch and rotate) can only be recognized on multi-touch enabled devices such as iOS or Android. You can extend the underlying C++ gesture recognition framework to support your custom gesture recognizers.

To enable the Gestures gem for your project, see Add modular features and assets with Gems (p. 1064).

**Topics**

- Configuring Gesture Recognizers (p. 1141)
- Responding to Gesture Input (p. 1142)

### Configuring Gesture Recognizers

You can configure gesture recognizers with C++ and the **Gestures** system component in the **System Entity Editor**. For more information, see Configuring Advanced Settings (p. 49).
To configure gesture recognizers

1. In the Project Configurator, click **Advanced Settings**.
2. In the **System Entity Editor**, click **Add Component**, and then expand **Gestures**.
3. For each gesture recognizer, specify your preferred values. For example, if you want to change the **Min Clicks or Taps** for a double press gesture, specify a new value.

   ![Image of gesture recognizer settings]

   **Note**
   
   Pause on the gesture recognizer to see a description and the valid values.

4. Make your changes and then click **Save**.

**Responding to Gesture Input**

Each gesture recognizer that the **Gestures** system component exposes corresponds to a gesture input channel that belongs to a gesture input device.

You can use gesture input channels in the same way as other input channels with C++, Lua, or Script Canvas. You can map the gesture input channels to gameplay actions with an **Input** component. This
component is part of the Input Management Framework (p. 1150) and Starting Point Input (p. 1206) gems.

To add input to an entity, see the Input (p. 642) component.

Example Lua Script

The following script listens for and responds to the default double press gesture.

```lua
function GestureExample:OnActivate()
    self.inputChannelNotificationBus = InputChannelNotificationBus.Connect(self);
end

function GestureExample:OnInputChannelEvent(inputChannel)
    if (inputChannel.channelName == InputDeviceGestures.gesture_double_press) then
        -- Respond to the default double press gesture
    end
end

function GestureExample:OnDeactivate()
    if (self.inputChannelNotificationBus) then
        self.inputChannelNotificationBus:Disconnect();
    end
end
```

For more information, see Input in Amazon Lumberyard (p. 1902).

Graphics Scripting

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Graphics Scripting gem enables graphics features in your scripts. With the gem enabled, you can use Script Canvas or Lua to write scripts to control features such as full screen effects, color correction, environment settings, shadow calculations, and more.

To use the Graphics Scripting gem, you must also enable the Script Canvas (p. 1201) gem. The LmbrCentral Gem (p. 1163) gem, which is also required, is enabled by default and includes material scripting controls and the High Quality Shadow (p. 614) component.

To enable gems, see Enabling Gems (p. 1064).

HMD Framework Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
HttpRequestor Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the HttpRequestor gem to make asynchronous HTTP/HTTPS requests and return data through a user-provided callback function. This gem uses the EBus for communication and provides all requests asynchronously. For more information, see Working with the Event Bus (EBus) system (p. 1851).

Note
This feature is supported only on Windows and replaces CryAction::HttpRequest.

Topics
- Getting Started (p. 1144)
- C++ API Using EBUS_EVENT (p. 1144)
- Example (p. 1146)

Getting Started

To use the HttpRequestor gem, you must enable it in your project.

To enable the HttpRequestor gem

1. In the Project Configurator (p. 43), select your project, and then click Enable Gems.
2. In the list of gems, select the check box next to HttpRequestor.
3. Click Save.
4. Enter the following command to build your project.

```bash
lmbr_waf configure
```

C++ API Using EBUS_EVENT

The HttpRequestor gem has separate sets of APIs for adding requests and adding text requests.

AddRequest, AddRequestWithHeaders, AddRequestWithHeadersAndBody

You can use the AddRequest, AddRequestWithHeaders, and AddRequestWithHeadersAndBody APIs to send generic HTTP requests to any website and receive the returned data in JSON format. The methods return the data received in the callback parameter.

Syntax

```c++
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddRequest, URI, method, callback)
```

```c++
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddRequestWithHeaders, URI, method, headers, callback)
```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, HttpStatusCode, URI, method, headers, body, callback)

Each add request method requires the URI, a method and a callback.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>AZStd::String</td>
<td>The fully qualified web address, in the following format: scheme:// [user:password@]host[:port]/]path[/query][#fragment]</td>
</tr>
<tr>
<td>method</td>
<td>Aws::Http::HttpMethod</td>
<td>The method type. The following values are supported: HTTP_GET, HTTP_POST, HTTP_DELETE, HTTP_PUT, HTTP_HEAD, and HTTP_PATCH.</td>
</tr>
<tr>
<td>callback</td>
<td></td>
<td>This function is called when the HTTP request is completed. The response body and code are present in the callback.</td>
</tr>
<tr>
<td>headers</td>
<td>HttpRequestor::Headers</td>
<td>The list of header fields for the HTTP request.</td>
</tr>
<tr>
<td>body</td>
<td>AZStd::String</td>
<td>Optional body to send with the request.</td>
</tr>
</tbody>
</table>

Return: No return value.

**JSON Request Callback**

This callback is returned for the AddRequest, AddRequestWithHeaders, and AddRequestWithHeadersAndBody methods.

```cpp
void Callback(const Aws::Utils::Json::JsonValue& json, Aws::Http::HttpResponseCode responseCode);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>json</td>
<td>Aws::Utils::Json::JsonValue</td>
<td>The JSON object. The life span of this object is valid only during the scope of the callback.</td>
</tr>
<tr>
<td>responseCode</td>
<td>Aws::Http::HttpResponseCode</td>
<td>The HTTP response code.</td>
</tr>
</tbody>
</table>

Return: No return value.

**AddTextRequest, AddTextRequestWithHeaders, AddTextRequestWithHeadersAndBody**

You can use the AddTextRequest, AddTextRequestWithHeaders, and AddTextRequestWithHeadersAndBody APIs to send a generic HTTP request to any website and receive the returned data in a text string. The methods return the data received in the callback parameter.
Syntax

EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddTextRequest, URI, method, callback)

EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddTextRequestWithHeaders, URI, method, headers, callback)

EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddTextRequestWithHeadersAndBody, URI, method, headers, body, callback)

Each add text request method requires the URI, a method and a callback.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>AZStd::String</td>
<td>The fully qualified web address, in the following format: scheme: [/user:password@]host[:port]/[/path][?query][#fragment]</td>
</tr>
<tr>
<td>method</td>
<td>Aws::Http::HttpMethod</td>
<td>The method type. The following values are supported: HTTP_GET, HTTP_POST, HTTP_DELETE, HTTP_PUT, HTTP_HEAD, and HTTP_PATCH.</td>
</tr>
<tr>
<td>callback</td>
<td></td>
<td>This function is called when the HTTP request is completed. The response body and code are present in the callback.</td>
</tr>
<tr>
<td>headers</td>
<td>HttpRequestor::Headers</td>
<td>The list of header fields for the HTTP request.</td>
</tr>
<tr>
<td>body</td>
<td>AZStd::String</td>
<td>Optional body to send with the request.</td>
</tr>
</tbody>
</table>

Return: No return value.

Text Request Callback

This callback is returned for the AddTextRequest, AddTextRequestWithHeaders, AddTextRequestWithHeadersAndBody methods.

void Callback(const AZStd::string& response, Aws::Http::HttpResponseCode responseCode);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>AZStd::string&amp;</td>
<td>The text returned from the server. The life span of this object is valid only during the scope of the callback.</td>
</tr>
<tr>
<td>responseCode</td>
<td>Aws::Http::HttpResponseCode</td>
<td>The HTTP response code.</td>
</tr>
</tbody>
</table>

Return: No return value.

Example

The following example uses the Metastream C++ API to obtain a gateway IP address.
Image Processing

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Image Processing gem enables the Texture Settings Editor, which you can use to customize and process image files and textures.

For more information, see Texture Settings Editor (p. 1734).

In-App Purchases Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The In-App Purchases Gem enables you to implement in-app purchases in your Android and iOS games. Platform-specific information is handled by the provided API, allowing you to use a single implementation of the In-App Purchases Gem for both Android and iOS.

Note
For Android, you must install the Google Play billing library from the Android SDK Manager. You can find this option under the extras section.

Handling Requests for Queries and Purchases

You can access the API through code by connecting to the EBus:

```
EBUS_EVENT(InAppPurchases::InAppPurchasesRequestBus, Initialize);
```

To use the API, include the InAppPurchasesBus.h header file.

You can also access the API through Script Canvas.
Initialize

Call this method first for all platforms. This method handles all necessary setup before the API can be used.

Parameters: None

QueryProductInfo

Use this method when the product IDs are shipped with the game. This method looks for a product ID file (product_ids.json on Android or product_ids.plist on iOS) and retrieves product details using the IDs that are specified in the file.

Parameters: None

QueryProductInfoByIds

Use this method to retrieve product details if the product IDs are provided at runtime, for example if they are retrieved from a server at runtime.

Parameters: AZStd::vector<AZStd::string>& productIds

QueryProductInfoById

Use this method to retrieve product details if the product IDs are provided at runtime, for example if they are retrieved from a server at runtime. Use this method to retrieve details for a single product only.

Parameters: AZStd::string& productId

QueryPurchasedProducts

Use this method to query the Google Play Store for products that were already purchased by the signed-in user. On iOS, this method reads the receipt that is stored on the device and lists the items purchased by the signed-in user.

Parameters: None

RestorePurchasedProducts

Use this method to restore purchases that were made by the user. This applies to purchases made on a different device, where the current device does not have receipts stored locally. This method is supported on iOS only.

Parameters: None

ConsumePurchase

Call this method for all purchases that are consumable, such as virtual currencies, health, and ammo. This method requires the purchase token provided by the Google Play Store when the product was purchased. This method is required and supported on Android only.

Parameters: AZStd::string& purchaseToken

FinishTransaction

Call this method at the end of a transaction. This method requires the transaction ID provided by the iOS App Store when the product was purchased. It also accepts a boolean parameter to indicate whether or not to download content that is hosted on Apple servers. If this method is not called, the transaction will be reported each time the game is restarted. This method is required and supported on iOS only.

Parameters: AZStd::string& transactionId | bool downloadHostedContent
**PurchaseProduct**

Use this method to request to purchase a product from the Google Play Store or the iOS App Store. This method requires the product ID of the product being purchased.

Parameters: AZStd::string& productId

**PurchaseProductWithDeveloperPayload**

Use this method to request to purchase a product from the Google Play Store or the iOS App Store. This method requires the product ID of the product being purchased. It accepts an additional parameter for the developer payload, which is used by Android to associate a purchase with a user account. When you request purchased products, you can use the developer payload to determine if the signed-in user made the purchase. On iOS, the user account is used in fraud detection.

Parameters: AZStd::string& productId | AZStd::string& developerPayload

**GetCachedProductInfo**

Use this method to return product details that are stored in a cache each time there is a query for information. The cache only stores product details that were retrieved during the previous call to QueryProductInfo/QueryProductInfoByIds/QueryProductInfoById.

Parameters: None

**GetCachedPurchasedProductInfo**

Use this method to return details for purchased products that are stored in a cache. The cache only stores details for purchased products that were retrieved during the previous call to QueryPurchasedProducts.

Parameters: None

**ClearCachedProductDetails**

Use this method to clear the product details that were cached by the previous call to query product details.

Parameters: None

**ClearCachedPurchasedProductDetails**

Use this method to clear the details for purchased products that were cached by the previous call to query details for purchased products.

Parameters: None

**Handling Responses to Queries and Purchases**

When a user makes a query or a purchase, the API sends the request to Apple or Google servers. Once a response is received, the API broadcasts the response on the InAppPurchasesResponse bus.

To handle responses to queries or purchases, overload the functions provided in the class in the InAppPurchasesResponseBus.h file.

**ProductInfoRetrieved**

This method is called when product information is retrieved for all requested products. Product information includes product ID, name, description, price, and more. Depending on the platform, the provided pointers must be cast to the appropriate type (ProductDetailsAndroid or ProductDetailsApple).
PurchasedProductsRetrieved

This method is called when details are retrieved for all purchased products. Details for purchased products include product ID, transaction ID, transaction time, and more. Depending on the platform, the provided pointers must be cast to the appropriate type (PurchasedProductDetailsAndroid or PurchasedProductDetailsApple).

Parameters:
```
const AZStd::vector<AZStd::unique_ptr<ProductDetails const>> & productDetails
```

NewProductPurchased

This method is called when a new product is successfully purchased.

Parameters:
```
const PurchasedProductDetails* purchasedProductDetails
```

PurchaseCancelled

This method is called when a purchase is canceled.

Parameters:
```
const PurchasedProductDetails* purchasedProductDetails
```

PurchaseRefunded

This method is called when a purchase is refunded.

Parameters:
```
const PurchasedProductDetails* purchasedProductDetails
```

PurchaseFailed

This method is called when a purchase fails.

Parameters:
```
const PurchasedProductDetails* purchasedProductDetails
```

HostedContentDownloadComplete

This method is called when content that is hosted on Apple servers is downloaded successfully. The transaction ID and download path are provided.

Parameters:
```
const AZStd::string& transactionId | AZStd::string& downloadedFileLocation
```

HostedContentDownloadFailed

This method is called when content that is hosted on Apple servers fails to download. The transaction ID and content ID of the failed content download are provided.

Parameters:
```
const AZStd::string& transactionId | const AZStd::string& contentId
```

Input Management Framework Gem

The Input Management Framework Gem is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The Input Management Framework gem provides a framework for managing cross-platform game input such as keyboard, controller, and touch in Lumberyard with the component entity system.

The gem works with the Starting Point Input (p. 1206) gem, which supplies the processed raw input. The Input Management Framework gem then takes that input and converts it to user-defined gameplay events with the Input (p. 642) component.

For an example, see Working with the Input Component (p. 643).

For more information, see Input in Amazon Lumberyard (p. 1902).

**Landscape Canvas Gem**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Landscape Canvas Gem provides the Landscape Canvas Editor; a node-based graph tool for authoring dynamic vegetation workflows to drive vegetation areas. This is a great improvement over the previous component based dynamic vegetation system because you can now manage an edit all of the inputs in one place, and easily visualize the data flow when you build dynamic vegetation. The Landscape Canvas Editor uses the same UI and UX principles as Script Canvas, making Landscape Canvas intuitive and familiar.

**Note**
Landscape Canvas is backwards compatible. Any Dynamic Vegetation content you have previously authored is automatically parsed and a graph is built for you from your existing levels.
Landscape Canvas has five node groups available that map to the components you previously would use to build dynamic vegetation: Gradient Modifiers, Gradients, Shapes, Utilities, and Vegetation Areas. The workflow is to create a new landscape canvas asset, and layout and connect nodes in the canvas to build a workflow for a vegetation area.

Enable the Landscape Canvas Gem

To make the Landscape Canvas Editor available in Lumberyard, you must build and configure your project with the Landscape Canvas Gem enabled.

1. Use Project Configurator to add the Landscape Canvas Gem to your project.

   Note
   The Landscape Canvas Gem requires the following Gems.
   - LmbrCentral
   - Vegetation
   - GraphCanvas
   - GraphModel
   - GradientSignal
   - SurfaceData

2. Configure your project.

   `lmbr_waf configure`

3. Build your project.

   `lmbr_waf build_win_x64_vs2019_profile -p all --progress`

For more information on Gems, see the Gems documentation (p. 1064).

Lightning Arc Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Lightning Arc Gem creates realistic electric arcing and sparking effects between points in a level.

While active, the entity sparks a new electrical arc to the assigned target entities randomly. The entity is able to trigger new sparks in either game mode or in AI/Physics mode.
Using the LightingArc Sample

The LightningArc Sample uses the LightningArc gem to demonstrate the various prescribed arc types.
Topics

- Enabling the Lightning Gem (p. 1154)
- Placing Lightning Arc (p. 1154)
- Configuring the Lightning Arc (p. 1155)
- Customizing a Lightning Arc Preset (p. 1156)

Enabling the Lightning Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You enable the Lightning gem from the Project Configurator. You must also enable the Game Effect Gem, as the Lightning Gem requires it. You can see the dependencies for gems in the Project Configurator. For more information, see Creating Lumberyard projects (p. 43).

To enable the Lightning gem, see Enabling Gems (p. 1064).

Placing Lightning Arc

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
When you place a lightning arc entity, you must specify at least one target. The lightning arcs between the lightning arc entity and each target that is linked. The lightning arc appears in Lumberyard Editor when you turn on AI/Physics or enter game mode (Ctrl + G).

**To place a lightning arc**

1. In the Rollup Bar, on the Object tab, click Entity.
2. Under Browser, expand Environment, and then drag LightningArc into your scene.
3. Under Entity Properties, ensure that Active is selected.
4. Click AI/Physics in the bottom toolbar. This makes the lightning arc visible in Lumberyard Editor after you place and link the targets.
5. To place one or more targets, in the Rollup Bar's Objects tab, click AI. Under Object Type, click Tagpoint.
6. Move your mouse into the scene, and click to place the tag point where your lightning will arc.
7. To link your tag point, select your lightning arc entity in the scene.
8. If necessary, scroll down or collapse other headings in the Rollup Bar to find Entity Links. Click Pick Target. Select the tag point you placed. Once it appears in the Link Name list, double-click the link name and change it to Target.
10. Expand materials\effects. Right-click the desired lightning effect. Then click Assign to Selected Objects. Close the Material Editor.

**Configuring the Lightning Arc**

You can configure the properties for the lightning arc entity to make the lightning arc show outside only, toggle wind effects, add delays and variations between arcs, and more. You can also carefully customize your lightning arcs by selecting different presets for the type of arc generated.

**To configure lightning arc entity parameters and properties**

1. In the Perspective viewport, select the lightning arc entity you want to configure.
2. Beneath Entity Params and Entity Properties, select or clear check boxes for the preferred effects.
Lightning Arc Entity Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Activates the effect</td>
</tr>
<tr>
<td>Render</td>
<td>ArcPreset Sets the specified arc preset defined in the lightningarceffects.xml file as explained in Customizing a Lightning Arc Preset (p. 1156).</td>
</tr>
<tr>
<td>Timing</td>
<td>Delay Sets the delay time between arcs</td>
</tr>
<tr>
<td></td>
<td>DelayVariation Sets the variation of the delay based on the delay time</td>
</tr>
</tbody>
</table>

Customizing a Lightning Arc Preset

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can customize your lightning arc entity using the presets in the lightningarceffects.xml file. You can also copy and modify existing presets to create your own customized lightning arc presets.

To use a lightning arc preset

1. In Lumberyard Editor, use the Select tool to select the lightning arc entity you want to customize.
2. In a text editor, open \dev\Gems\LightningArc\Assets\libs\lightningarc \lightningarceffects.xml in the Lumberyard root directory (lumberyard_version\dev).
3. Choose one of the existing presets from the lightningarceffects.xml file (follows Arc name in the example) and, in Lumberyard Editor, under Entity Properties, enter your chosen Arc name into the ArcPreset field.

For example, enter ExtendedArc or KickSparks, which are existing names of presets as shown in the following lightningarceffects.xml file. This sample shows only the partial contents; open the file on your computer to view the full contents of the file.

```xml
<LightningArc>
  <Arc name="Default">
    <param name="lightningDeviation" value="0.2" />
    <param name="lightningFuzzyness" value="0.1" />
    <param name="branchMaxLevel" value="1" />
    <param name="branchProbability" value="2.0" />
    <param name="lightningVelocity" value="0.6" />
    <param name="strikeTimeMin" value="0.35" />
    <param name="strikeTimeMax" value="0.35" />
    <param name="strikeFadeOut" value="0.6" />
    <param name="strikeNumSegments" value="6" />
    <param name="strikeNumPoints" value="5" />
    <param name="maxNumStrikes" value="6" />
    <param name="beamSize" value="0.2" />
    <param name="beamTexTiling" value="0.25" />
    <param name="beamTexShift" value="0.05" />
  </Arc>
</LightningArc>
```
To create a new lightning arc preset

1. Open the lightningarceffects.xml file.
2. Copy the text (between and including <Arc name="Name"> through </Arc>) for an existing preset.
3. Paste it at the end of the file before the </LightningArc> closing bracket.
4. Replace the Arc name with your own custom preset name, then modify the following parameters to fit your needs.

The following table lists definitions for the parameters in the lightningarceffects.xml file.

**Lightning Arc Entity Properties**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightningDeviation</td>
<td>The smoothness of the effect in meters.</td>
</tr>
<tr>
<td>lightningFuzzyness</td>
<td>The noisiness of the effect in meters.</td>
</tr>
</tbody>
</table>

Version 1.28

1157
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>branchMaxLevel</td>
<td>Allows child branches to strike out of the main beam and child branches to strike out from other child beams if this value is 2 or higher. A setting of 0 or 1 is recommended.</td>
</tr>
<tr>
<td>branchProbability</td>
<td>Probability that child branch will strike out from another beam segment. Consider these examples:</td>
</tr>
<tr>
<td></td>
<td>• 0 – No branch is generated</td>
</tr>
<tr>
<td></td>
<td>• 0.5 – Creates one branch per beam half the time</td>
</tr>
<tr>
<td></td>
<td>• 1.0 – Creates one branch per beam</td>
</tr>
<tr>
<td></td>
<td>• 2.0 – Creates 2 branches per beam</td>
</tr>
<tr>
<td>lightningVelocity</td>
<td>Rate at which a branch shifts upward from its original position after being triggered.</td>
</tr>
<tr>
<td>strikeTimeMin</td>
<td>Minimum time a branch remains visible.</td>
</tr>
<tr>
<td>strikeTimeMax</td>
<td>Maximum time a branch remains visible.</td>
</tr>
<tr>
<td>strikeFadeOut</td>
<td>Time to fade out after a branch disappears. This setting decreases the branch beamSize to 0 instead of actually fading with transparency.</td>
</tr>
<tr>
<td>strikeNumSegments</td>
<td>Number of snaking segments generated.</td>
</tr>
<tr>
<td>strikeNumPoints</td>
<td>Number of points per segment generated to create the noisy effect.</td>
</tr>
<tr>
<td></td>
<td>The number of actual segments generated is defined by strikeNumSegments * strikeNumPoints.</td>
</tr>
<tr>
<td></td>
<td>When the code generates the geometry, it creates a camera-aligned beam with exactly two triangles. This means the number of triangles per strike is strikeNumSegments<em>strikeNumPoint</em>2. Since maxNumStrikes is the hard limit of potential number of sparks active at any time, the potential number polygons of a given lightning effect is strikeNumSegments<em>strikeNumPoint</em>2*maxNumStrike.</td>
</tr>
<tr>
<td></td>
<td>Note that with the LightningArc entity, each lightning strike triggers a new lightning strike. Therefore the total poly count of a given effect can be much higher. The game has internal limits for the total amount of lightning effects, lightning strikes, and polygons that cannot be surpassed.</td>
</tr>
<tr>
<td>maxNumStrikes</td>
<td>Hard limit on the number of beam segments that can be generated.</td>
</tr>
<tr>
<td>beamSize</td>
<td>Width of the beam generated. Child beams have half the width.</td>
</tr>
<tr>
<td>beamTexTiling</td>
<td>Texture tiling depends on the world size. A value of 2.0 means the texture wraps around twice every meter. A value of 0.25 means the texture will wrap around every 4 meters.</td>
</tr>
<tr>
<td>beamTexShift</td>
<td>Rate at which the U coordinate moves in a given direction. While beamTexTiling affects only the U coordinate, the V coordinate is automatically calculated to select one of the texture's frames.</td>
</tr>
<tr>
<td>beamTexFrames</td>
<td>Number of frames in the animation.</td>
</tr>
<tr>
<td>beamTexFPS</td>
<td>Frames per second of the multiframe animation.</td>
</tr>
</tbody>
</table>
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Legacy Terrain Gem enables the legacy terrain system in Lumberyard version 1.24 or later. With this gem, you can use the Legacy Terrain level component in the Level Inspector to add terrain to levels. Levels that were created in Lumberyard versions before 1.24 that use the legacy terrain system display a warning when they are loaded in Lumberyard version 1.24 or later, and the new Legacy Terrain level component is automatically added to the level. Newly created levels have the Legacy Terrain level component added by default.

Note
In Lumberyard versions 1.24 and later, the Legacy Terrain Gem is enabled by default for Lumberyard projects and is required to enable the legacy terrain system. For more information about Gems, see the Gems documentation (p. 1064).

The Legacy Terrain Gem doesn't introduce new tools to edit terrain. Terrain texture layers are created with the Terrain Texture Layers editor. Terrain heightmaps are painted with the Terrain Editor. Terrain can be painted and sculpted in the Perspective viewport with the Terrain Tool. For more information about editing terrain, see Creating Terrain (p. 1220).

In addition to the Legacy Terrain Gem, Lumberyard version 1.24 introduces a new terrain API: AzFramework::Terrain::TerrainDataRequestBus (p. 1160). The legacy terrain system has been refactored to integrate the new terrain API. This new API makes it easier to seamlessly replace the legacy terrain system with your own terrain system. The legacy terrain APIs that are present in I3DEngine.h and ITerrain.h have been marked for deprecation. If your project uses the legacy terrain APIs, you should migrate your code as soon as possible to use the new AzFramework::Terrain::TerrainDataRequestBus. The legacy terrain APIs will be removed in a future release.

Disable terrain editor tools
Lumberyard builds the terrain editor tools by default. If the Legacy Terrain Gem is disabled, a WAF configure-time warning is raised because terrain can’t be added to the level, so the terrain editor tools serve no purpose.

If you are not using the Legacy Terrain Gem, you can disable the terrain editor tools in the WAF build settings. To disable the terrain editor tools, use one of the following options.

- Manually set the build flag enable_legacy_terrain_editor to False in lumberyard_version\dev\_WAF\user_settings.options:

  [Build Options]
  ...
  enable_legacy_terrain_editor = False
  ...

Version 1.28
1159
• Change the value `enable_legacy_terrain_editor` in the Lumberyard WAF options dialog.

1. Open the Lumberyard WAF options from `lumberyard_version\dev` with the following command.

   ```bash
   lmbr_waf show_option_dialog
   ```

2. Find the `enable_legacy_terrain_editor` option in the Build tab and deselect the check box on the right to disable the terrain editor tools.

Legacy terrain system

• Legacy Terrain level component (p. 651)
• TerrainDataRequestBus API (p. 1160)

TerrainDataRequestBus API

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

TerrainDataRequestBus is an API introduced in Lumberyard release 1.24. This API presents an EBus interface to query a terrain system. TerrainDataRequestBus enables you to easily swap different implementations of terrain systems. The Legacy Terrain level component is implemented with the TerrainDataRequestBus API.

The TerrainDataRequestBus is a single address, single handler EBus. This means that only one component that provides the service TerrainService can be active at any given time.
The Legacy Terrain level component (p. 651), which is provided by the Legacy Terrain Gem (p. 1159), is a TerrainService provider and can answer to calls made to the TerrainDataRequestBus.

The TerrainDataRequestBus header file is located here: lumberyard_version/Code/Framework/AzFramework/AzFramework/Terrain/TerrainDataRequestBus.h.

The TerrainDataRequestBus API replaces all terrain-related APIs that historically have been available in lumberyard_version/Code/CryEngine/CryCommon/I3DEngine.h and lumberyard_version/Code/CryEngine/CryCommon/ITerrain.h. The old APIs have been marked for deprecation, along with notes about the new replacement API. The example below is from I3DEngine.h:

```cpp
I3DEngine.h GetTerrainElevation()
```

```cpp
//! LUMBERYARD_DEPRECATED(LY-107351) Use AzFramework::Terrain::TerrainDataRequestBus::GetHeight*(Sampler::BILINEAR) instead.
// Summary:
//     Gets the interpolated terrain elevation for a specified location.
// Notes:
//     All x,y values are valid.
// Arguments:
//     x - X coordinate of the location
//     y - Y coordinate of the location
// Return Value:
//     A float which indicate the elevation level.
virtual float GetTerrainElevation(float x, float y, int nSID = DEFAULT_SID) = 0;
```

Notes on thread safety

TerrainDataRequestBus was designed for multi-threaded usage. The following are some recommendations to efficiently use the TerrainDataRequestBus API with thread safety in mind.

For loops running outside of the Main thread

Unsafe and Fast (Bad)

In the following code, the Legacy Terrain level component is created and destroyed on the Main thread when switching levels or entering/leaving Game Mode.

```cpp
auto terrain = AzFramework::Terrain::TerrainDataRequestBus::FindFirstHandler();
for ()
{
    float height = terrain->GetHeightFromFloats(x,y);
    //Do something.
}
```

Safe and Slow (Better)

The following code locks and unlocks its internal mutex each time a call is made on a thread-safe EBus.

```cpp
```
for (){
    float height = 0.0f;
    AzFramework::Terrain::TerrainDataRequestBus::Broadcast(height,
        &AzFramework::Terrain::TerrainDataRequest::GetHeightFromFloats,x,y,
        AzFramework::Terrain::TerrainDataRequest::Sampler::BILINEAR);
    //Do something.
}

Safe and Fast (Best)

In the following code, EnumerateHandlers locks the EBus mutex only once, allowing you to run
the code you need, and EnumerateHandlers unlock the EBus mutex when the task completes.

auto enumerationCallback = [&](AzFramework::Terrain::TerrainDataRequests* terrain) ->
    bool
    {
        float height = terrain->GetHeightFromFloats(x,y);
        //Do something.
        // Only one handler should exist.
        return false;
    }
AzFramework::Terrain::TerrainDataRequestBus::EnumerateHandlers(enumerationCallback);

A lock-less alternative

Assuming that you have control over when to start or stop threads/jobs, the lock-less alternative
would be to subclass TerrainDataNotificationBus, and stop/restart your threads/jobs by
reacting accordingly to one of these callbacks:

AzFramework::Terrain::TerrainDataNotificationBus

virtual void OnTerrainDataCreateBegin() {};
virtual void OnTerrainDataCreateEnd() {};
virtual void OnTerrainDataDestroyBegin() {};
virtual void OnTerrainDataDestroyEnd() {};

Fast and safe, assuming proper usage of TerrainDataNotificationBus

Assuming threads/jobs are never running while the legacy terrain system is being destroyed or
created, then the following case is safe and fast:

auto terrain = AzFramework::Terrain::TerrainDataRequestBus::FindFirstHandler();
for (){
    float height = terrain->GetHeightFromFloats(x,y);
    //Do something.
}
LmbrCentral Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The LmbrCentral gem contains required code and assets for running Lumberyard features, such as animation, audio, Lumberyard Editor, rendering, scripting, and so on.

**Note**
The LmbrCentral gem is required and enabled by default for Lumberyard projects.

LyShine

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The LyShine gem enables Lumberyard's in-game UI system. With the UI system, you can create dynamic user interfaces for your games.

**Note**
The LyShine gem is required and enabled by default in new Lumberyard projects.

For more information, see Creating and Customizing Project User Interfaces (p. 2874).

Maestro Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The Maestro gem enables Lumberyard’s cinematic sequence system. With the cinematic sequence system, you can create cutscenes for your games with the Track View and render them as a series of frames and/or a movie in the editor and during gameplay.

**Note**
The Maestro gem is required and enabled by default in new Lumberyard projects.

For more information, see [Create cinematic sequences](p. 1583).

## Metastream Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Twitch Metastream is a feature that allows broadcasters to customize their streams with overlays of statistics and events from their game session. Using any web authoring tool, such as Dreamweaver or CoffeeCup, broadcasters can create custom HTML5 pages to control the information, graphics, layout, and behavior of each unique overlay. With Metastream, broadcasters can create more polished, interactive viewing experiences on any of their favorite streaming services, similar to what you see in professional Esports and TV broadcasts.

Examples of information displayed in an overlay include:

- Character art
- Character strengths and weaknesses
- Player standings
- Stats for two leaders in a match
- Gold collected
- Kills, deaths, and assists
- Damage dealt

Broadcasters can switch between different graphic overlays that are timed to game events. They can also use a picture-in-picture style to display complementary information such as a minimap and live team stats.

To enable broadcasters to use Twitch Metastream, you must do the following:

1. Enable the Metastream gem in your project.
2. Add a single line of code for each event you want broadcasters to access.

**Note**
Twitch Metastream is supported on Windows only.
Adding the Metastream Gem

Enable the Metastream gem in your project to turn on the local HTTP Metastream server that is included with Lumberyard.

To enable the Metastream gem, see Enabling Gems (p. 1064).

Setting Options for the HTTP Server

After you enable Metastream, an HTTP server is embedded into the game client and serves as the access point for exposed data. You can set the following options for the HTTP server.

`metastream_enabled`

Read-only console variable (CVAR) that describes the current state of the embedded HTTP server. 0 = disabled. 1 = enabled.

`metastream_serverOptions`

Sets the options for the embedded HTTP server. Options are a set of semicolon separated key=value pairs.

If a key value requires a semi-colon character ‘;’ or the equal character ‘=’ you may use the $semi macro or $equ.

The embedded HTTP server is based on CivetWeb. You can find the full list of options that can be set at:


For security reasons, the following CivetWeb options are ignored:

- `enable_directory_listing`
- `cgi_interpreter`
- `run_as_user`
- `put_delete_auth_file`

Files that you place inside the document root will be served by the HTTP server. See Accessing Data through the HTTP API (p. 1170).

You can also use the following console commands:

`metastream_start`

Starts the embedded HTTP server

`metastream_stop`

Stops the embedded HTTP server

Exposing Data through Metastream

Metastream exposes data through the C++ API.

C++ API

Controlling the HTTP Server

The Metastream Gem uses the following API to start the HTTP server:
bool result; // True when the server is successfully started, false when an error occurs.
Metastream::MetastreamRequestBus::BroadcastResult(result,&Metastream::MetastreamRequests::StartHTTPServer);

Attempting to start the HTTP server when it is already running has no effect, and the call returns true.

The Metastream Gem uses the following API to stop the HTTP server:

Metastream::MetastreamRequestBus::Broadcast(&Metastream::MetastreamRequests::StopHTTPServer);

If the server is not running, attempting to stop the server has no effect.

**Exposing Data**

The Metastream Gem uses a simple API to expose in-game data using the EBus system. See Accessing Data through the HTTP API (p. 1170) for these values. Currently, the Metastream API supports UTF8 strings, bools, Vec3, doubles, signed and unsigned 64-bit values. The EBus system requires these calls to be discrete. This API allows you to add values to objects and/or arrays and the cache. This allows for a fully flexible system in exposing data to a Web client.

**Add to Cache**

To add an object to the root cache, use the following syntax:

```cpp
void MetastreamRequests::AddStringToCache(const char* table, const char* key, const char* value)
void MetastreamRequests::AddBoolToCache(const char* table, const char* key, bool value)
void MetastreamRequests::AddVec3ToCache(const char* table, const char* key, const Vec3& value)
void MetastreamRequests::AddDoubleToCache(const char* table, const char* key, double value)
void MetastreamRequests::AddUnsigned64ToCache(const char* table, const char* key, AZ::u64 value)
void MetastreamRequests::AddSigned64ToCache(const char* table, const char* key, AZ::s64 value)
void MetastreamRequests::AddArrayToCache(const char* table, const char* key, const char* arrayName)
void MetastreamRequests::AddObjectToCache(const char* table, const char* key, const char* objectName)
```

Definitions for the parameters listed:

- `table`
  - Name of the table.
- `key`
  - Name of the key.
- `value`
  - The value to add. If the value exists, it is updated. If the value type is bool, it is represented in the JSON doc as true or false. Signed, unsigned, and double types are represented as JSON numbers; strings are UTF8 and will be escaped, if necessary.
- `arrayName`
  - The name of the array to add. If no array exists, then empty array is added. The array is deleted after it is added to the cache.
- `objectName`
  - The name of the object to add. If no object exists, then a NULL object is added. The object is deleted after it is added to the cache.
None of the above returns any values.

Add to Array

To add an object to an array, use the following syntax:

```cpp
void MetastreamRequests::AddStringToArray(const char* table, const char* arrayName, const char* value)
void MetastreamRequests::AddBoolToArray(const char* table, const char* arrayName, bool value)
void MetastreamRequests::AddVec3ToArray(const char* table, const char* arrayName, const Vec3 & value)
void MetastreamRequests::AddDoubleToArray(const char* table, const char* arrayName, double value)
void MetastreamRequests::AddUnsigned64ToArray(const char* table, const char* arrayName, AZ::u64 value)
void MetastreamRequests::AddSigned64ToArray(const char* table, const char* arrayName, AZ::s64 value)
void MetastreamRequests::AddObjectToArray(const char* table, const char* destArrayName, const char* sourceObjectName)
```

Definitions for the parameters listed:

- **table**
  Name of the table.

- **arrayName**
  The name of the array to add. If no array exists, then an empty array is added. The array is deleted after it is added to the cache.

- **value**
  The value to add to the array. If value type is bool, it is represented in the JSON doc as `true` or `false`. Signed, unsigned, and double types are represented as JSON numbers; strings are UTF8 and will be escaped, if necessary.

- **arrayName**
  The name of the array to add to. If no array exists, one is created.

- **destArrayName**
  The name of destination array to add to. If no array exists, one is created.

- **sourceObjectName**
  The name of the object to add to the array. If no object exists, then a NULL object is added. The object is deleted after it is added to the cache.

None of the above returns any values.

Add to Object

To add an object to an object, use the following syntax:

```cpp
void MetastreamRequests::AddArrayToObject(const char* table, const char* destObjectName, const char* key, const char* srcArrayName)
void MetastreamRequests::AddObjectToObject(const char* table, const char* destObjectName, const char* key, const char* sourceObjectName)
void MetastreamRequests::AddStringToObject(const char* table, const char* objectName, const char* key, const char* value)
```
void MetastreamRequests::AddBoolToObject(const char* table, const char* objectName, const char* key, bool value)
void MetastreamRequests::AddVec3ToObject(const char* table, const char* objectName, const char* key, const Vec3 & value)
void MetastreamRequests::AddDoubleToObject(const char* table, const char* objectName, const char* key, double value)
void MetastreamRequests::AddUnsigned64ToObject(const char* table, const char* objectName, const char* key, AZ::u64 value)
void MetastreamRequests::AddSigned64ToObject(const char* table, const char* objectName, const char* key, AZ::s64 value)

Definitions for the parameters listed:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>key</td>
<td>Name of the key.</td>
</tr>
<tr>
<td>objectName</td>
<td>The name of the object to add. If no object exists, then one is created.</td>
</tr>
<tr>
<td>value</td>
<td>The value to add. If the value exists, it is updated. If value type is bool, it is represented in the JSON doc as true or false. Signed, unsigned, and double types are represented as JSON numbers; strings are UTF8 and will be escaped, if necessary.</td>
</tr>
<tr>
<td>srcArrayName</td>
<td>The name of the array to add to. If no array exists, one is created. The array is deleted after it is added to the object.</td>
</tr>
<tr>
<td>sourceObjectName</td>
<td>The name of the object to add. If no object exists, then a NULL object is added. The object is deleted after it is added to the cache.</td>
</tr>
</tbody>
</table>

None of the above returns any values.

Examples

The following example shows how to use the Metastream C++ API in a project:

```
Metastream::MetastreamRequestBus::Broadcast(&Metastream::MetastreamRequestBus::Events::AddToCache, table, key, value);
```

Note

Any value that is added to the cache should be JSON compliant. For information, see the JSON RFC.

The following example shows how to reflect the system information. Basically all of the info is added to a object name, sysInfo, and this object is then added to the cache as systeminfo.

Compound object:

```
Metastream::MetastreamRequestBus::Broadcast(
```
Metastream::MetastreamRequestBus::Events::AddUnsigned64ToObject,
kDataBaseName.c_str(), "sysInfo", "drivespace", GetFreeDriveSpace());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddUnsigned64ToObject,
    kDataBaseName.c_str(), "sysInfo", "memoryload", GetMemoryLoad());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddDoubleToObject,
    kDataBaseName.c_str(), "sysInfo", "cpuloadsystem", GetCPULoadSystem());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddDoubleToObject,
    kDataBaseName.c_str(), "sysInfo", "cpuloadprocess", GetCPUloadProcess());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddUnsigned64ToObject,
    kDataBaseName.c_str(), "sysInfo", "tickcount", GetTickCount64());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddObjectToCache,
    kDataBaseName.c_str(), "systeminfo", "sysInfo");

---

**Metastream Lua Bindings**

As of Lumberyard 1.10, Twitch Metastream reflects using the behavior context, which enables you to use Metastream through Script Canvas and Lua.

The following methods can be invoked from Lua:

```
-- Controlling the HTTP server:
MetastreamRequestBus::Broadcast.StartHTTPServer();
MetastreamRequestBus::Broadcast.StopHTTPServer();

-- Adding to the root cache directly:
MetastreamRequestBus::Broadcast>AddStringToCache(table, key, value);  -- where value is a string
MetastreamRequestBus::Broadcast>AddBoolToCache(table, key, value);     -- where value is a bool
MetastreamRequestBus::Broadcast>AddDoubleToCache(table, key, value);   -- where value is a double
MetastreamRequestBus::Broadcast>AddUnsigned64ToCache(table, key, value);  -- where value is an unsigned 64-bit number
MetastreamRequestBus::Broadcast>AddSigned64ToCache(table, key, value);  -- where value is a signed 64-bit number
MetastreamRequestBus::Broadcast>AddArrayToCache(table, key, arrayName);  -- where arrayName is the name of a temporary array (see below)
MetastreamRequestBus::Broadcast>AddObjectToCache(table, key, objectName); -- where objectName is the name of a temporary object (see below)

-- Adding to a temporary Array:
MetastreamRequestBus::Broadcast>AddStringToArray(table, arrayName, value);
MetastreamRequestBus::Broadcast>AddBoolToArray(table, arrayName, value);
MetastreamRequestBus::Broadcast>AddDoubleToArray(table, arrayName, value);
MetastreamRequestBus::Broadcast>AddUnsigned64ToArray(table, arrayName, value);
MetastreamRequestBus::Broadcast>AddSigned64ToArray(table, arrayName, value);
MetastreamRequestBus::Broadcast>AddObjectToArray(table, arrayName, objectName);

-- Adding to a temporary Object:
MetastreamRequestBus::Broadcast>AddStringToObject(table, objectName, key, value);
MetastreamRequestBus::Broadcast>AddBoolToObject(table, objectName, key, value);
MetastreamRequestBus::Broadcast>AddDoubleToObject(table, objectName, key, value);
MetastreamRequestBus::Broadcast>AddUnsigned64ToObject(table, objectName, key, value);
```
MetastreamRequestBus.Broadcast.AddSigned64ToObject(table, objectName, key, value);
MetastreamRequestBus.Broadcast.AddArrayToObject(table, objectName, key, arrayName);
MetastreamRequestBus.Broadcast.AddObjectToObject(table, objectName, key, objectName);

Accessing Data through the HTTP API

You can access game data that has been exposed through Metastream by using the HTTP API Get requests. You can then use JavaScript to work with the data.

http://localhost:port/pathToFile

Serves a file from the document root. File types include HTML, JS, CSS, images, sounds, resources, or assets.

**Note**
The data path is reserved for Metastream data. Files that are saved to the `document_root/data/` directory will not be accessible.

http://localhost:port/data

Returns a list of available Metastream tables that contain `key=value` pairs.

http://localhost:port/data?table=table_name

Returns a list of all Metastream keys in the specified table.

**Note**
You can retrieve multiple key-value pairs in a single request by listing the keys in a comma-separated list. For example, `http://localhost:8082/data?table=sample&key=key1,key2,key3`

http://localhost:port/data?table=table_name&key=key_name

Returns the value for the specified key in the specified table.

**Note**
Multiple `key=value` pairs can be retrieved in a single request by listing the desired keys separated by commas. For example, `http://localhost:8082/data?table=sample&key=key1,key2,key3`.

To list all keys and their values for a table: `http://localhost:8082/data?table=sample&key=*`

Data requests are returned in the following format:

<table>
<thead>
<tr>
<th>Request</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>/data</td>
<td>{ &quot;tables&quot;: [ &quot;table1&quot;, &quot;table2&quot;, ... ] }</td>
</tr>
<tr>
<td>/data?table=table_name</td>
<td>{ &quot;keys&quot;: [ &quot;key1&quot;, &quot;key2&quot;, ... ] }</td>
</tr>
<tr>
<td>/data?table=table_name&amp;key=key_name</td>
<td>{ &quot;key_name&quot;: value }</td>
</tr>
<tr>
<td>/data?table=table_name&amp;key=keys_list</td>
<td>{ &quot;key1&quot;: value1, &quot;key2&quot;: value2, ... }</td>
</tr>
</tbody>
</table>
Microphone Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Microphone Gem is a dependency for the Cloud Canvas Speech Recognition Gem (p. 1122). This means that when you enable the Cloud Canvas Speech Recognition Gem in the Project Configurator, the Microphone Gem is also automatically added.

The Microphone Gem's API is located at \dev\Code\CryEngine\CryCommon\MicrophoneBus.h.

This gem connects to a hardware recording device and enables you to capture an audio signal. Before running the game or editor, set the default recording device in your Operating System. Check your levels and adjust the gain accordingly. Once set up, the Microphone Gem connects to that device at application startup. Capturing starts when you start a capturing session and stops when you end the capturing session.

Clients using the mic data are expected to consume the data at a pace that meets or exceeds the data rate of the microphone. If consumption rate is too slow, the captured data can lag progressively behind real-time.

When consuming samples, you can specify a desired output format configuration, and the output is automatically converted.

Note the following:

- Only Windows is currently supported.
- Mono and stereo configurations only are supported for microphone input and client output.
- Only linear PCM samples are supported. This can be in either IEEE float (32-bit) or 16-bit signed integer formats. Typically mics will capture float samples.
- Hardware configuration changes, such as changing the default recording device, aren't handled at run time.
- Choosing a device to connect to is not yet supported. The Microphone gem connects only to the default device.
- Only a single consumer should be used. This is because consuming samples from the Microphone gem advances the read position of internal buffers containing the mic data. If two clients attempt to consume the data, each client obtains only portions of the original data.

Multiplayer Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Multiplayer Gem makes it easy for you to add some basic network functionality to your games without digging too deeply into specific details or customization. To make this possible, Lumberyard has created a Multiplayer Lobby component and CVars for general networking use.

You can also use the Multiplayer sample project (p. 155) for an example of how to build and structure multiplayer games that use the various features of the GridMate networking (p. 1990) library.

For more information about networking in Lumberyard, see Using Lumberyard Networking (p. 1990).
The Multiplayer Lobby Component

You can use the self-contained lobby from Lumberyard's component entity system and add the `MultiplayerLobbyComponent` to a component entity in a scene. The `MultiplayerLobbyComponent` provides a basic lobby that can perform the following tasks:

- Search for an active game session
- Create a visual list of game sessions
- Join a particular game session
- Create a game session
- Name a game session
- Determine the map to load into
- Report errors

Supported Session Services

The `MultiplayerLobbyComponent` supports all of Lumberyard's session services, including `LANSessionService` and `GameLiftSessionService`.

Configuration Settings

The `MultiplayerLobbyComponent` has configuration settings that you can use to customize the hosted sessions that the component creates.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Players</td>
<td>The maximum number of players allowed to join the session.</td>
</tr>
<tr>
<td>Port</td>
<td>The port on which the game session operates.</td>
</tr>
<tr>
<td>Enable Disconnect Detection</td>
<td>Enables disconnect detection. If a player's connection does not respond to inquiries from the session host within the specified timeout window, the player is disconnected from the session.</td>
</tr>
<tr>
<td>Timeout</td>
<td>The timeout window, in milliseconds, that a client has to respond to inquiries from the session host before being disconnected, if disconnect detection is enabled.</td>
</tr>
<tr>
<td>Default Map</td>
<td>The default value for the Map field of the display.</td>
</tr>
<tr>
<td>Default Server Name</td>
<td>The default value for the Server Name field of the display.</td>
</tr>
<tr>
<td>Default Matchmaking Config</td>
<td>The default value for the Matchmaking Config to use for GameLift.</td>
</tr>
</tbody>
</table>

Note

This component does not handle the initialization of encryption, but utilizes encryption if it is already enabled. If you want to use encryption with the component, you must configure encryption before you use the component.
LAN Lobby Description

The following image shows a sample LAN lobby and its details.

1. Return – Exits the current lobby and returns to the SessionService selection screen.
2. Session List – Populates with server results from a successful GridSearch. This lists all of the sessions available, the name of each session, and the number of players in each session.
3. Refresh – Performs a GridSearch on the selected SessionService and displays the results in the session list.
4. Join – Attempts to join the currently selected session in the session list. If an error occurs, an error message is displayed. If no GridSession has been selected, this option is disabled.
5. Create a Server – Use these text boxes to configure sessions.
   - Server Name – Specifies the name that will be displayed in the session list when GridSession instances are searched for and the created GridSession is returned.
   - Map – Use to specify the map to be loaded when the GridSession is successfully created.
6. Create Server – Attempts to create a GridSession in the selected SessionService.

GameLift Lobby Description

The following image shows a sample GameLift lobby and its details.
1. **Return** – Exits the current lobby and returns to the SessionService selection screen.

2. **Create Server / Join Server / FlexMatch** – Radio buttons to toggle the details window for the selected radio button. (Current selection = Create Server).

3. Use these text boxes to configure sessions:
   - **ServerName** – Specifies the name that will be displayed in the session list when GridSession instances are searched for and the created GridSession is returned.
   - **Map Name** – Use to specify the map to be loaded when the GridSession is successfully created.

4. Need to fill only one of the below:
   - **Queue Name** – Queue to use to create the server game session (First priority).
   - **Fleet Id** – Fleet id to use to create the server game session (Second priority).
   - **Alias Id** – Alias id to use to create the server game session (Second priority).

5. **Create Server** – Attempts to create a GameLiftSession in the selected GameLift fleet or queue.
1. Need to fill only one of the below:
   - **Queue Name** – Queue to use to join a server game session (First priority).
   - **Fleet Id** – Fleet id to use to join a server game session (Second priority).
   - **Alias Id** – Alias id to use to join a server game session (Second priority).

2. **SessionList** – Populates with server results from a successful GridSearch. This lists all of the sessions available, the name of each session, and the number of players in each session.

3. **Refresh** – Performs a search for available game sessions using queue, fleet or alias. Displays the results as list in session list.

4. **Join** – Attempts to join the currently selected session in the session list. If an error occurs, an error message is displayed. If no GridSession has been selected, this option is disabled.
1. **Config Name** – Matchmaking configuration to use to start GameLift matchmaking.
2. **Start Matchmaking** – Will call GameLift service to try and place the player in available game session based on the matchmaking config. On success, player joins the game session.

**Multiplayer Gem CVars**

For convenience, the Multiplayer Gem provides the following console variable commands that you can use to test features or configurations.

**Client CVars**

**mpphost <port number>**

Creates a GridSession using the LANSessionService on the local machine. The optional port number determines the port on which the grid session runs.

**mpjoin <server address> <port number>**

Performs a GridSearch using the LANSessionService and joins the first session returned. The server address and port number are optional parameters that can be used to refine the GridSearch.

**mpsearch**

Performs a GridSearch using the LANSessionService.

**mpdisconnect**

Disconnects the local client from its active GridSession, if any.

**gm_version**

Sets the GridMate version number.
**gm_netsec_enable**

Specifies whether network level encryption is used for connections. A value of 0 disables encryption and a value of 1 enables encryption. The default for the multiplayer gem is 0 (disabled).

**Note**

In the Multiplayer Sample, this value is overridden to be enabled in `game.cfg`.

**gm_netsec_private_key**

Specifies the private key file (PEM format) for a secure network connection when `FileDataSource` is used as a certificate manager. This setting overrides any previously configured values in `FileDataSource`. If network encryption is enabled, this value must be set on the host of a `GridSession` when the session is created. This setting is disabled in release builds.

**gm_netsec_certificate**

Specifies the private key file (PEM format) for a secure network connection when `FileDataSource` is used as a certificate manager. This setting overrides any previously configured values in `FileDataSource`. If network encryption is enabled, this value must be set on the host of a `GridSession` when the session is created. This setting is disabled in release builds.

**gm_netsec_ca**

Specifies the CA Certificate file (PEM format) for a secure network connection when the `FileDataSource` is used as a certificate manager. This setting overrides any previously configured values in `FileDataSource`. This value must be set on the client when trying to join a `GridSession` using encryption. This setting is disabled in release builds.

**gm_netsec_verify_client**

Enables client verification. If not enabled, only the server is authenticated. Must be set on the host of a `GridSession`. The default is 0, disabled.

**gm_disconnectDetection**

Sets whether disconnect detection is enabled for a particular `GridSession`. Must be set on the host of `GridSession`. If disconnect detection is enabled, clients that do not respond to inquiries within the timeout window are disconnected. The default is 1, enabled.

**gm_disconnectDetectionRttThreshold**

A RTT threshold, in milliseconds, that specifies the time after which a connection is dropped when there is no response. The default is 500.

**gm_disconnectDetectionPacketLossThreshold**

A value between 0 and 1 that represents the percentage of packets that are permitted to be lost in communication with a particular client. If a client experiences a packet loss greater than the specified value, the connection is dropped. The default is 0.3.

**gm_recvPacketsLimit**

The maximum number of packets per second allowed to be received from an existing connection. The default is 0 (no limit).

**gm_ipversion**

Specifies which Internet protocol version to use. Possible values are `IPv4` or `IPv6`. The default is IPv4.

**gm_securityData**

Specifies the security data for session.
gm_replicasSendTime

Specifies, in milliseconds, the time interval between replica updates. A value of 0 binds the interval to the GridMate tick rate. The default is 0.

gm_replicasSendLimit

Limits the amount of replica data sent per second. 0 specifies no limit. The default is 0. This value is disabled in release builds.

gm_burstTimeLimit

Specifies, in seconds, an interval window during which bandwidth capping is not applied. If bandwidth usage continues to exceed the replica send limit outside of this interval, the sent amount is capped at the send limit. The default is 10. This variable is disabled in release builds.

gamelift_fleet_id

Specifies the GameLift FleetId to use for this client when starting GameLiftSessionService.

gamelift_aws_access_key

Specifies the AWS access key associated with the GameLift fleet.

gamelift_aws_secret_key

Specifies the AWS secret key associated with the GameLift fleet.

gamelift_aws_region

Specifies the AWS region associated with the GameLift fleet. The default is us-west-2.

gamelift_endpoint

Specifies the GameLift service endpoint. The default is gamelift.us-west-2.amazonaws.com.

gamelift_alias_id

Specifies the GameLift alias ID to use with the client.

gamelift_player_id

Specifies the GameLift player ID associated with the client.

gamelift_stop_client

Stops the GameLiftSessionService and terminates the current GridSession, if one is connected.

gamelift_queue_name

Specifies GameLift queue to use for this client when creating or joining a game session.

gamelift_matchmaking_config_name

Specifies GameLift FlexMatch matchmaking configuration to use for this client when being placed in a game session.

sv_port <local_port>

Sets the local UDP port that initializes the socket. The default port is 30090. To use the ephemeral port, set the port to 0. This is useful if you want to connect to a server on the same computer as the client.

Server CVars

gamelift_flexmatch_enable

Configures the server process to enable custom matchmaking backfill for FlexMatch.
Multiplayer Analytics Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Multiplayer Analytics Gem is an additional GridMate Gem for analyzing your network traffic live, using an in-game overlay.

When enabled, you can press the HOME key to bring up an in-game UI. The following example is from the Multiplayer Sample included with Lumberyard.

For more information about how to use this Gem in Lumberyard, see Using Multiplayer Analytics (p. 2092).
NullVR Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the NullVR gem to run your level through the head-mounted displays (HMD) framework without a connected VR device. This gem is useful for VR graphics debugging.

The NullVR Gem has a lower priority than other VR gems and their devices. That means that if any other VR gem is enabled and its appropriate VR device is connected, then the HMD framework uses that VR gem. The NullVR Gem is enabled only if no other VR gem is enabled or no VR device is connected.

When Lumberyard is using the NullVR Gem, the following appears in the console:

```
[HMD][Null] - Null Device
```

Console Variables

The NullVR Gem, when enabled, makes certain console variables available. These console variables take effect only if the NullVR Gem is in use; that is, no other VR gems are enabled or no other VR devices are connected.

**Note**
The default values in the following list came from Vive at the time of this writing.

- `hmd_null_aspectRatio`
  Aspect ratio of the null VR implementation.
  Default: 0.9

- `hmd_null_eyeOffsetX`
  Eye X offset (horizontal distance).
  Default: -0.0346999988

- `hmd_null_fov`
  The field of view in radians. Can be updated at runtime.
  Default: 1.91986

- `hmd_null_fovH`
  The field of view height in radians. Used on start.
  Default: 1.84633982

- `hmd_null_fovV`
  The field of view width in radians. Used on start.
  Default: 1.94590104

- `hmd_null_frustumPlane_horizontalDistance`
  Frustum plane horizontal distance.
  Default: -0.0701720715
hmd_null_frustumPlane_verticalDistance

Frustum plane vertical distance.

Default: -0.00206005573

hmd_null_renderHeight

The height of the render texture in pixels. Set once on start.

Default: 1680

hmd_null_renderWidth

The width of the render texture in pixels. Set once on start.

Default: 1512

NVIDIA Blast gem

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The NVIDIA Blast gem uses the NVIDIA Blast library to provide fast, high-fidelity destruction simulation in Amazon Lumberyard.

Note

NVIDIA Blast for Lumberyard requires a SideFX Houdini commercial or indie license to create assets. The apprentice license is not sufficient. For more information on Houdini, see SideFX’s home page.

The precompiled Houdini plug-ins supplied with the NVIDIA Blast gem require Houdini 18.0.

For NVIDIA Blast developer information, see Simulated destruction with NVIDIA Blast (p. 2833).

Contents

- Functionality provided by the NVIDIA Blast gem (p. 1181)
- Enable the NVIDIA Blast gem (p. 1182)

Functionality provided by the NVIDIA Blast gem

The NVIDIA Blast gem provides the following:

- **Blast Family Mesh Data** component that adds NVIDIA Blast meshes to an entity.
- **Blast Family** component that enables NVIDIA Blast simulation for an entity.
- **Blast Configuration** editor available in the Tools menu in Lumberyard Editor.
- **Blast Materials** to set physical properties for NVIDIA Blast assets available in Asset Editor.
- **Blast Script Canvas** nodes to script destruction simulation.
- Plug-ins and Houdini Digital Assets for SideFX Houdini to fracture geometry and export NVIDIA Blast assets.
- A Python Asset Builder to process NVIDIA Blast assets and generate blast slices.
• A public C++ API that allows other systems and gems to access NVIDIA Blast simulation functionality.

Enable the NVIDIA Blast gem

To enable the NVIDIA Blast gem, do the following:

1. Use Project Configurator (p. 44) to add the NVIDIA Blast gem to your project. The NVIDIA Blast gem requires the following gems as dependencies:
   • LmbrCentral
   • PhysX

   **Important**
   Though not required, we highly recommend that you enable the Python Asset Builder gem (p. 1192) with the NVIDIA Blast gem. The NVIDIA Blast gem includes a Python asset builder script that automatically processes mesh assets for NVIDIA Blast and creates a blast slice asset.

2. Configure your project. Use the following command.
   
   ```text
   lmbr_waf configure
   ```

3. Build your project. Use the following command.

   ```text
   lmbr_waf build_win_x64_vs2019_profile -p all --progress
   ```

   For more information on gems, see the Gems system documentation (p. 1064).

**NVIDIA Cloth gem**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Physical cloth simulations can create more immersive environments and characters. The NVIDIA Cloth gem uses the NVIDIA Cloth library to provide fast, robust cloth simulation in Amazon Lumberyard.

For information on using NVIDIA Cloth, see Simulate cloth with NVIDIA Cloth (p. 2857).

**Functionality provided by the NVIDIA Cloth gem**

The NVIDIA Cloth gem provides the following:

• **Cloth** modifier for Mesh and Actor import in FBX Settings.
• **Cloth** component for entities that contain a Mesh or Actor component.
• Cloth colliders that can be added to actors in Animation Editor.
• Mesh and Actor example assets and slices located in: `lumberyard_version\dev\Gems\NvCloth\Assets`
• A public C++ API that allows other systems and gems to access cloth simulation functionality.
Enable the NVIDIA Cloth gem

To enable the NVIDIA Cloth gem, do the following:

**Note**
By default, NVIDIA Cloth simulation is performed on the CPU. To enable GPU accelerated simulation for NVIDIA Cloth, follow the installation instructions here: [GPU cloth simulation with NVIDIA CUDA](p. 2871)

1. Use Project Configurator (p. 44) to add the NVIDIA Cloth gem to your project. The NVIDIA Cloth gem requires the following gems as dependencies:
   - LmbrCentral
   - Emotion FX Animation
2. Configure your project. Use the following command.

   ```
   lmbr_waf configure
   ```
3. Build your project. Use the following command.

   ```
   lmbr_waf build_win_x64_vs2019_profile -p all --progress
   ```

For more information on gems, see the Gems system documentation (p. 1064).

Physically Based Shaders (PBS) Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Physically Based Shader (PBS) Reference Gem is a simple collection of 32 example materials that you can apply to a Mesh (p. 684) component. They demonstrate a wide range of surface material types that make use of the Illum Shader (p. 1697). These materials range from matte to shiny and metallic or reflective. For more information about rendering physically based materials, see Shader Rendering System (p. 1639).

The PBS Reference Gem is enabled by default in new projects and is also enabled in the default Samples Project. You use the Material Editor to preview, modify, and apply a material to a mesh.

The following procedure describes a very simple example work flow for applying a PBS Reference Gem material to an object. This procedure assumes your project is Samples Project (the default project for Lumberyard).

To apply materials contained in PBS Reference Gem

1. Open a level in Lumberyard.
2. Create a new entity (p. 463).
3. Add (p. 479) a Mesh (p. 684) component to your entity.
4. In the Entity Inspector (p. 475), in the Mesh component, next to the Mesh Asset property, click ‘...’ to specify an asset.

   In the Pick Static Mesh dialog, browse to \SamplesProject\Objects\Primitives. Pick one of the .cfg files in that directory. Click OK.
The object that you picked now appears in your Viewport.
5. Make sure your object is still selected.

Open the Material Editor by pressing M or by navigating to Tools, Material Editor.

6. In the Material Editor's navigation pane, browse to \Gems\PBSreferenceMaterials\Assets\materials\pbs_reference.

Select one of the .mtl files listed.
7. In the Material Editor's toolbar, click the Assign Item to Selected Object icon.
Close or move the **Material Editor** to see your **Viewport**.

The assigned material is now applied to your object.
PhysX

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard's PhysX system acts upon entities to create realistic physical effects such as collision detection and rigid body dynamics simulation. Enable the PhysX gem to use the PhysX system for your game project.

For more information, see Simulating physics behavior with the PhysX system (p. 2772).

PhysX Characters

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The PhysX Characters gem provides physics for your characters in your game. The gem integrates with the EMotion FX Animation Editor system.

Note
To enable the PhysX Characters gem, you must first enable the PhysX (p. 1188) gem.

The PhysX Characters gem adds the following components:

- PhysX Character Controller (p. 729)
- PhysX Ragdoll (p. 757)

For more information, see PhysX system (p. 2772).

PhysX Debug

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The PhysX Debug gem provides features to debug visualizations for your PhysX scene geometry, such as the PhysX Collider (p. 755), PhysX Rigid Body (p. 759) component, and so on.
When you enter console variables or the use the ImGui tool, you can view the PhysX debug lines in editor and game modes. This gem uses data directly from PhysX to show a culled (limited by proximity to the camera) view of the simulated world in real time.

In editor mode, this gem displays PhysX shapes within a given distance of the viewport camera. In game mode, this gem uses the currently active camera to visualize a culled view of the the PhysX scene.

This gem includes the following features:

- Visualize debug rendering of physics geometry, such as collision primitives, terrain, shapes, and forces.
- Control mechanisms using console variables, the PhysX settings menu, and the ImGui tool.
- Visualization frustum culling.
- PhysX visual debugger hooks and controls using the third-party tool visual debugger.
- Proximity based debug visualizations of collision meshes.

**Note**
This feature is similar to the legacy p_draw_helpers=1 console variable.

To enable the PhysX Debug gem, you must first enable the PhysX (p. 1188) and ImGui gems.

For more information, see Debugging PhysX (p. 2826).

### Physics Entities Gem

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Physics Entities Gem is a collection of physics entities used to simulate physical events such as explosions, gravity fields, or wind, or to physicalize objects such as cloth, breakable entities, or ropes. Physical entities that are related to a body instead of an event are connected to an object.

**To access the Physics entities**

1. In the Rollup Bar, on the Objects tab, choose Entity.
2. Expand Physics.
3. Drag the entity into your level in the viewport.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Primitive Assets Gem provides primitive objects that are physics enabled and unit size (1x1x1), with a gray checkerboard texture. The checkerboard texture is RGB middle gray (127,127,127) and sRGB middle gray (187,187,187). Diffuse, normal, and specular maps are included. The primitive object models have import settings, a material, and a slice. Each slice contains the object mesh and the appropriate Shape and Rigid Body Physics components. You can use these to add the cube, sphere, and cylinder objects to your level and manipulate the objects, create a placeholder, or test Lumberyard Editor features.

The asset files for the primitive objects are in the `\lumberyard_version\dev\Gems\PrimitiveAssets\Assets` directory.

You can add the primitive objects to your game project by doing either of the following:

- Create a new project from the Default template in the Project Configurator. By default the Primitive Assets Gem is enabled for projects that are created from this template.
- Enable the Primitive Assets Gem for your existing project in the Project Configurator.

For more information, see Creating a Game Project in Lumberyard (p. 45).
Process Life Management Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Process Life Management gem demonstrates how you can respond to various application lifecycle events dispatched by the Lumberyard engine, in order to pause your game, display a modal splash screen, or anything else you may need to do when your application loses/regains focus.

You can access all system-specific events from C++ (even without enabling the Process Life Management gem) by connecting to the appropriate EBus. Lumberyard also generates platform-agnostic events so that you can handle these events for all supported platforms.

Lumberyard Application Lifecycle Events

<table>
<thead>
<tr>
<th>Lumberyard Application Lifecycle Events</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnApplicationConstrained</td>
<td>applicationWillResignActive</td>
<td>onPause()</td>
</tr>
<tr>
<td>OnApplicationUnconstrained</td>
<td>applicationDidBecomeActive</td>
<td>onResume()</td>
</tr>
<tr>
<td>OnApplicationSuspended</td>
<td>applicationDidEnterBackground</td>
<td>onPause()</td>
</tr>
<tr>
<td>OnApplicationResumed</td>
<td>applicationWillEnterForeground</td>
<td>onResume()</td>
</tr>
<tr>
<td>OnMobileApplicationWillTerminate</td>
<td>applicationWillTerminate</td>
<td>onDestroy()</td>
</tr>
<tr>
<td>OnMobileApplicationLowMemoryWarning</td>
<td>applicationDidReceiveMemoryWarning</td>
<td>onLowMemory()</td>
</tr>
</tbody>
</table>

As demonstrated in ProcessLifeManagementGem.h\ProcessLifeManagementGem.cpp, use the following basic steps to receive process lifecycle events in your game.
To receive process lifecycle events in your game

1. Derive your class from `AzFramework::ApplicationLifecycleEvents::Bus::Handler` (or `AzFramework::[Ios|Android|Windows]LifecycleEvents::Bus::Handler` for platform specific events).
2. Override the functions corresponding to the events that you want to override:

   ```
   void OnApplicationConstrained(Event /lastEvent/) override;
   void OnApplicationUnconstrained(Event /lastEvent/) override;
   void OnApplicationSuspended(Event /lastEvent/) override;
   void OnApplicationResumed(Event /lastEvent/) override;
   ```
3. Connect to the event bus when you want to start listening for events. In addition, be sure to disconnect when you no longer want to receive them. Use the following syntax:

   ```
   ApplicationLifecycleEvents::Bus::Handler::BusConnect();
   ...
   ApplicationLifecycleEvents::Bus::Handler::BusDisconnect();
   ```

Python Asset Builder gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With **Python Asset Builder**, you can create Python scripts that process custom assets that are produced from content creation tools such as Maya and Houdini, or any content tool with a known file format.

For information on using **Python Asset Builder**, see Process custom assets with Python Asset Builder (p. 448).

Enable the Python Asset Builder gem

To enable the Python Asset Builder gem, do the following:

1. Use **Project Configurator** (p. 44) to enable the **Python Asset Builder** gem in your project.
2. Configure your project. Use the following command.

   ```
   lmbr_waf configure
   ```
3. Build your project. Use the following command.

   ```
   lmbr_waf build_win_x64_vs2019_profile -p all --progress
   ```

For more information on gems, see the Gems system documentation (p. 1064).

RAD Telemetry Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
RAD Telemetry 3 is an instrumentation-based profiling and performance visualization middleware product created by RAD Game Tools. The RAD Telemetry Gem provides a RAD Telemetry 3 integration for Lumberyard for those who have licensed RAD Telemetry.

The RAD Telemetry Gem provides one example of how to integrate instrumentation-based profiling middleware. You could write a gem for your own instrumentation-based profiler and leverage Lumberyard’s existing performance markers. See the Profiler.h file in the /dev/Code/Framework/AzCore/AzCore/Debug directory to see how the RAD Telemetry Gem ties into Lumberyard’s existing performance markers.

**Note**
If the RAD Telemetry Gem is disabled or you do not have the RAD redistributable files for a particular platform, the performance instrumentation compiles to Lumberyard’s performance markers.

**To enable the RAD Telemetry Gem**

1. Extract the Telemetry redistributables from RAD Game Tools to /Gems/RADTelemetry/External. You should see the following subdirectories, which match the subdirectories provided by RAD:
   - Dll
   - Lib
   - Include
   - Docs

   **Note**
   - If you are compiling for case-sensitive platforms, note that the first letter of each subdirectory listed above is capitalized.
   - When compiling a project with the RAD Telemetry Gem enabled, the static lib file for the target platform must exist in the Gems/RADTelemetry/External/ directory. The Waf build system then compiles with the AZ_PROFILE_TELEMETRY defined globally and links RAD Telemetry libraries for the specified platform. You can add additional platforms by editing the rad_telemetry.json file (located in the Gems/RADTelemetry/3rdParty directory).

2. Extract RAD Telemetry Tools to the /Gems/RADTelemetry/Tools directory. This should include telemetry.exe and your license file from RAD.

3. Use the Project Configurator to enable (p. 1064) the RAD Telemetry Gem for your project. You must enable this gem (and any other gem) for each project. Gems are not globally enabled.

4. Build your project (p. 61). You must build using profile (preferred) or debug (p. 61) in order to enable RAD Telemetry.

**Instrumenting Your Code**

With the RAD Telemetry Gem, Lumberyard introduces a set of scoped performance markers:

- **AZ_PROFILE_FUNCTION** – Instruments entire functions. Automatically names the performance event with the function's name.

- **AZ_PROFILE_SCOPE** – Instruments a local scope within a function of interest. You must provide the name.

- **AZ_PROFILE_SCOPE_DYNAMIC** – Instruments a printf style format string to dynamically generate a performance event name. Use the dynamic name sparingly because there may be performance overhead of a string copy and transmission over the network.
A scoped performance marker constructs an object that calls a start event and calls a stop event when it is destroyed. This means that you do not have to worry about early returns. We recommend that you use the AZ_PROFILE events when marking up your code, as it allows you to switch to Driller for a different view of profiling data.

Lumberyard also uses the following legacy performance event markers:

- PROFILE_FUNCTION
- PROFILE_FRAME

Capturing with RAD Telemetry

See the RAD Telemetry documentation for more information. The basic procedure is described below.

To capture with RAD Telemetry

1. Set the console variables for the RAD Telemetry capture using either a configuration file or by entering them into the console at runtime.

   All RAD Telemetry console variables start with radtm_. For example, radtm_Address and radtm_Port if your Telemetry server is on another machine. The console variables are defined in the RADTelemetryModule.cpp file. You can augment these console variables to better suit profiling for your title.

2. Use either telemetry.exe or the standalone telemetry server to start your RAD Telemetry server.

3. Run radtm_ToggleEnabled to begin a capture. You can also set this from the command line when launching your application with +radtm_ToggleEnabled.

4. Run radtm_ToggleEnabled again from the console or the remote console, or shut down your game process to end the session. As currently instrumented, captures can get large quickly, so you may want to perform focused captures around performance scenarios of interest (under 30 seconds in length).

5. Use the Telemetry visualizer (telemetry.exe) to analyze your capture.

Rain Gem

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Rain Gem creates realistic rain effects in your levels, including rain drops, puddles, mist, wet surfaces, and splashes. To enable the Rain Gem in your project, see Add modular features and assets with Gems (p. 1064).

This gem is a game object extension. On initialization, it preloads all textures listed in the raintextures.xml file.
Place only a single Rain entity in your scene.

Placing Rain

You can place rain and customize it for your level by modifying properties for amount of puddles, strength and frequency of puddle ripples, quantity of rain, size and speed of the rain drops, and more.

To add rain to your level

1. In the Rollup Bar, click Entity.
2. Under Browser, expand Environment.
3. Drag the Rain entity into your scene.

Configuring Rain

You can configure the rain's properties under Entity Params and Entity Properties.

### Rain Entity Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>Sets overall amount of the rain entity's various effects</td>
</tr>
<tr>
<td>DiffuseDarkening</td>
<td>Sets the degree to which the rain darkens the surface diffuse</td>
</tr>
<tr>
<td>DisableOcclusion</td>
<td>Turns off checking whether an object is under cover and should be occluded from rain</td>
</tr>
<tr>
<td>Enabled</td>
<td>Toggles the rain effect</td>
</tr>
<tr>
<td>IgnoreVisareas</td>
<td>Continue to render rain when player is inside a vis area</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PuddlesAmount</td>
<td>Sets the size and number of puddles that the rain creates</td>
</tr>
<tr>
<td>PuddlesMaskAmount</td>
<td>Sets the strength of the puddle mask to balance different puddle results</td>
</tr>
<tr>
<td>PuddlesRippleAmount</td>
<td>Sets the height and frequency of ripples in rain puddles</td>
</tr>
<tr>
<td>Radius</td>
<td>Sets the area on which rain falls</td>
</tr>
<tr>
<td>RainDropsAmount</td>
<td>Sets the number of rain drops</td>
</tr>
<tr>
<td>RainDropsLighting</td>
<td>Sets the brightness of the rain drops</td>
</tr>
<tr>
<td>RainDropsSpeed</td>
<td>Sets the rate at which rain falls</td>
</tr>
<tr>
<td>SplashesAmount</td>
<td>Sets the degree of splashing on a surface</td>
</tr>
</tbody>
</table>

### Using Console Variables for Rain

You can use the following console variables for the rain entity.

**Rain Entity Console Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_Rain</td>
<td>Enables rain rendering</td>
</tr>
<tr>
<td>r_RainAmount</td>
<td>Sets rain amount</td>
</tr>
<tr>
<td>r_RainDistMultiplier</td>
<td>Multiplier for the rain layer's distance from the camera</td>
</tr>
<tr>
<td>r_RainDropsEffect</td>
<td>Enables rain drops effect</td>
</tr>
<tr>
<td>r_RainIgnoreNearest</td>
<td>Disables the layer showing the reflection of objects in rainy or wet areas</td>
</tr>
<tr>
<td>r_RainMaxViewDist</td>
<td>Sets the maximum distance at which rain is visible</td>
</tr>
<tr>
<td>r_RainMaxViewDist_Deferred</td>
<td>Maximum distance (in meters) at which the deferred rain reflection layer is</td>
</tr>
<tr>
<td>r_RainOccluderSize</td>
<td>Used to block rain for objects bigger than this value</td>
</tr>
</tbody>
</table>

### Using the Rain Sample

The Rain Sample uses the Rain, Clouds, and LightningArc gems to demonstrate how to use rain as an environment special effects (FX) in a level. The Lightning entity (from the LightningArc gem) shows how the lightning FX can enhance a rain storm with flashes of light and random strikes of lightning on the ground. The clouds are enabled to show how they can fill a scene.
Render to Texture

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Enable the Render to Texture gem for your game so that you can use the Render to Texture (p. 781) component. You can use this component to render the scene to a texture from the viewport of the selected camera. For example, you can add game objects such as mirrors, security camera screens, and draw 3D models.

Note
The Render to Texture component currently supports DirectX 11 for Windows.
SaveData Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The SaveData gem encapsulates all the functionality in Lumberyard for saving game and individual user data under a single platform-agnostic set of API operations.

To enable gems, see Enabling Gems (p. 1064).

The SaveData gem uses the Event Bus (EBus) (p. 1851), Lumberyard's general-purpose communication system for dispatching notifications and receiving requests. To make requests related to saving or loading persistent user data, use the SaveData gem's SaveDataRequests bus. To listen for notifications related to saving persistent user data, use the SaveDataNotifications bus.

Making Requests to Save Data

When making requests to save or load data using the SaveDataRequestBus, keep the following points in mind:

- The SaveData gem is responsible only for saving and loading generic data buffers. Your game must serialize or deserialize data using a data format, such as JSON or XML, that you deem appropriate. However, convenience functions are provided that save or load an object that has been reflected using an AZ::SerializeContext (p. 1033).
- Each save data buffer must be uniquely identified by a string. On most operating systems and devices, this string is the name of the file to which the data buffer is written.

Saving Data for Local User IDs

SaveData gem communications that deal with local user profiles depend on the local user ID that uniquely identifies a user on the local device. When using local user IDs, keep in mind the following points:

- A save data buffer can optionally be associated with a local user ID.
- Data buffers that have the same name can be stored separately for each local user ID.
- When you call the save data functions, you must either specify a local user ID or pass in the value AzFramework::LocalUserIdNone. This requirement helps you to consider whether the data that you are saving should be associated with a user.
- Data associated with a local user ID is saved into a container or directory that is unique to the local user ID and the application.
- Data not associated with a local user ID is saved into a 'global' container or directory unique to the application.

For more information about the SaveDataRequests bus, see the commented source code at lumberyard_version\dev\Gems\SaveData\Code\Include\SaveData\SaveDataRequestBus.h.
Getting Save Data Notifications

All save and load operations performed by the SaveData gem are asynchronous. Therefore, you must either subscribe to receive Save Data notifications or supply a callback function that notifies you when a save or load operation completes. This action is always performed in the main thread.

For more information about the SaveDataNotifications bus, see the commented source code at `lumberyard_version\dev\Gems\SaveData\Code\Include\SaveData\SaveDataNotificationBus.h`.

Save Data Code Example

The following example code uses the SaveData gem to save and load buffers and objects to and from persistent storage.

```cpp
const AZ::u64 testSaveDataSize = 9;
const char* testSaveDataName = "TestSaveData";
char testSaveData[testSaveDataSize] = {'a', 'b', 'c', '1', '2', '3', 'x', 'y', 'z'};

void SaveBufferToPersistentStorage()
{
    SaveData::SaveDataRequests::SaveDataBufferParams params;
    params.dataBuffer.reset(testSaveData);
    params.dataBufferSize = testSaveDataSize;
    params.dataBufferName = testSaveDataName;
    params.callback = [](const SaveData::SaveDataNotifications::DataBufferSavedParams& onSavedParams)
    {
        if (onSavedParams.result != SaveData::SaveDataNotifications::Result::Success)
        {
            // Error handling
        }
    };
    SaveData::SaveDataRequestBus::Broadcast(&SaveData::SaveDataRequests::SaveDataBuffer, params);
}

void LoadBufferFromPersistentStorage()
{
    SaveData::SaveDataRequests::LoadDataBufferParams params;
    params.dataBufferName = testSaveDataName;
    params.callback = [](const SaveData::SaveDataNotifications::DataBufferLoadedParams& onLoadedParams)
    {
        if (onLoadedParams.result == SaveData::SaveDataNotifications::Result::Success)
        {
            // SaveDataNotifications::DataBuffer is a shared_ptr, so you can choose to either preserve the buffer (by keeping a reference to it), or just let it go out of scope so it will be deleted.
            SaveDataNotifications::DataBuffer loadedDataBuffer = onLoadedParams.dataBuffer;
            // Use the loaded data buffer...
        }
        else
        {
            // Error handling
        }
    };
    SaveData::SaveDataRequestBus::Broadcast(&SaveData::SaveDataRequests::LoadDataBuffer, params);
}
```
class TestObject
{
public:
    AZ_RTTI(TestObject, "{9CE29971-8FE2-41FF-AD5B-CB15F1B92834}");
    static void Reflect(AZ::SerializeContext& sc)
    {
        sc.Class<TestObject>()
            ->Version(1)
            ->Field("testString", &TestObject::testString)
            ->Field("testFloat", &TestObject::testFloat)
            ->Field("testInt", &TestObject::testInt)
            ->Field("testBool", &TestObject::testBool)
            ;
    }
    AZStd::string testString;
    float testFloat = 0.0f;
    int testInt = 0;
    bool testBool = false;
};

void SaveObjectToPersistentStorage()
{
    // Reflect the test object class (if not already done).
    AZ::SerializeContext serializeContext;
    TestObject::Reflect(serializeContext);

    // Create a test object instance to save.
    AZStd::shared_ptr<TestObject> testObject = AZStd::make_shared<TestObject>();

    // Setup the save data params
    SaveData::SaveDataRequests::SaveOrLoadObjectParams<TestObject> params;
    params.serializableObject = testObject;
    params.serializeContext = &serializeContext; // Omit to use the global
    params.dataBufferName = "TestSaveObject";
    params.callback = [](const
        SaveData::SaveDataRequests::SaveOrLoadObjectParams<TestObject>& callbackParams,
        SaveData::SaveDataNotifications::Result callbackResult)
        {
            if (callbackResult != SaveData::SaveDataNotifications::Result::Success)
            {
                // Error handling
            }
        };
    SaveData::SaveDataRequests::SaveObject(params);
}

void LoadObjectFromPersistentStorage(const AzFramework::LocalUserId& localUserId = AzFramework::LocalUserIdNone)
{
    // Reflect the test object class (if not already done).
    AZ::SerializeContext serializeContext;
    TestObject::Reflect(serializeContext);

    // Create a test object to load.
    AZStd::shared_ptr<TestObject> testObject = AZStd::make_shared<TestObject>();

    // Setup the load data params
    SaveData::SaveDataRequests::SaveOrLoadObjectParams<TestObject> params;
    params.serializableObject = testObject;
    params.serializeContext = &serializeContext; // Omit to use the global
    params.dataBufferName = "TestSaveObject";
    params.callback = [](const
        SaveData::SaveDataRequests::SaveOrLoadObjectParams<TestObject>& callbackParams,
        SaveData::SaveDataNotifications::Result callbackResult)
{  
    if (onLoadedParams.result == SaveData::SaveDataNotifications::Result::Success)  
    {  
        // Use the loaded data buffer...  
        callbackParams.serializableObject;  
    }  
    else  
    {  
        // Error handling  
    }  
};  
SaveData::SaveDataRequests::LoadObject(params);

Scene Processing Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Scene Processing gem to specify the default settings for processing scene files such as .fbx for your games. For example, you can use the FBX Settings tool to specify the file types and settings that you want for your .fbx files. Asset Processor then processes the files so that they are available in the Asset Browser.

Note
The Scene Processing gem is required and enabled by default for Lumberyard projects.

For information about working with file types and settings, see Customize FBX asset export with FBX Settings (p. 409).

Script Canvas Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Script Canvas gem enables Lumberyard's visual scripting environment. With the gem enabled, you can use the Script Canvas editor to create game logic without needing to write your own code.

To enable gems, see Enabling Gems (p. 1064).
For more information about working with Script Canvas graphs, see Creating Gameplay with Script Canvas (p. 2420).

**Scripted Entity Tweener Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Scripted Entity Tweener gem animates entities in Lumberyard and is particularly useful and powerful for your game UI entities. For more information, see Scripted Tweener Entity System (p. 3011).

**Sky Clouds Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Sky Clouds gem creates realistic and detailed cloud and weather effects in your game levels. You can create clouds with either simple, sprite-based shading, or more complex, voxelized 3D volume shading.

To enable the Sky Clouds gem for your project, see Add modular features and assets with Gems (p. 1064).

For more information about creating and placing clouds in your level, see Sky Cloud (p. 830).

For more information about working with clouds, including setting cloud shading parameters, adding 3D cloud shadows, and creating 3D cloud templates, see Adding Clouds (p. 1293).

**Example**
Slice Favorites

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With the **Slice Favorites** gem, you can flag a slice as a favorite in the **Asset Browser**.

After you create a slice favorite, you can easily access and instantiate it into your scenes from the **Entity Outliner**.

For more information, see **Slice Favorites** (p. 526).

Snow Gem

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the **CryEntity Removal gem** using the **Project Configurator** or the **command line**. To learn more about legacy features, see the **Lumberyard Legacy Reference**.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Snow Gem creates realistic snow effects in your levels, including snowflake and surface effects, such as snow buildup. To enable the Snow Gem in your project, see **Add modular features and assets with Gems** (p. 1064).

**Note**

Place only a single **Snow** entity in your scene.
Placing Snow

You can place your snow and customize it to your level by modifying properties for brightness, gravity, size and quantity of snow flakes, how much snow and frost builds on a surface, and more.

To add snow to your level

1. In the Rollup Bar, click Entity.
2. Under Browser, expand Environment.
3. Drag the Snow entity into your scene.

Configuring Snow

You can configure the snow's properties under Entity Params and Entity Properties.

Snow Entity Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables snow effect</td>
</tr>
<tr>
<td>Radius</td>
<td>Sets the area on which snow falls</td>
</tr>
<tr>
<td>SnowFall</td>
<td></td>
</tr>
<tr>
<td>Brightness</td>
<td>Sets the brightness of the snow effect</td>
</tr>
<tr>
<td>GravityScale</td>
<td>Sets the gravity strength, which determines the rate at which snow falls</td>
</tr>
<tr>
<td>SnowFlakeCount</td>
<td>Sets the quantity of snowflakes</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SnowFlakeSize</td>
<td>Sets size of individual snowflakes</td>
</tr>
<tr>
<td>TurbulenceFreq</td>
<td>Sets the frequency of the turbulence affecting the snow</td>
</tr>
<tr>
<td>TurbulenceStrength</td>
<td>Sets the strength of the turbulence affecting the snow</td>
</tr>
<tr>
<td>WindScale</td>
<td>Determines the impact of wind on the falling snow</td>
</tr>
<tr>
<td><em>Surface</em></td>
<td></td>
</tr>
<tr>
<td>FrostAmount</td>
<td>Sets the amount of frost on a surface</td>
</tr>
<tr>
<td>SnowAmount</td>
<td>Sets the amount of snow on a surface</td>
</tr>
<tr>
<td>SurfaceFreezing</td>
<td>Sets the degree to which surfaces appear frozen</td>
</tr>
</tbody>
</table>

### Using Console Variables for Snow

You can use the following console variables [console variables](p. 210) for the snow entity.

#### Snow Entity Console Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_Snow</td>
<td>Enables snow rendering</td>
</tr>
<tr>
<td>r_SnowDisplacement</td>
<td>Enables displacement for snow accumulation</td>
</tr>
<tr>
<td>r_SnowFlakeClusters</td>
<td>Number of snow flake clusters</td>
</tr>
<tr>
<td>r_SnowHalfRes</td>
<td>When enabled, renders snow at half resolution to conserve fill rate</td>
</tr>
</tbody>
</table>

### Using the Snow Sample

The Snow Sample uses the Snow and Clouds gems to demonstrate how to use the Snow entity as an environment special effects in a level. The Snow entity shows how snow falls and provides properties that you can set to randomly change the snow fall over time, creating a more dynamic weather experience.
Starting Point Input Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Starting Point Input gem works with the Input Management Framework (p. 1150) gem to map low-level input events to high-level gameplay actions. For example, you can specify in your game that pressing the keyboard spacebar results in a gameplay action, such as jump.

For information about working with input events, see the Input (p. 642) component.

For information about working with input gestures such as touch, swipe, and rotate, see the Gestures (p. 1141) gem.
Twitch v5 Gem

The Twitch v5 Gem is deprecated and will be removed in a future version of Lumberyard. It has been replaced by the Twitch API Gem.

The Twitch v5 Gem is a wrapper for the Legacy Twitch v5 RESTful API. Install this Gem into your Lumberyard project to access legacy Twitch API v5 features.

Warning
Twitch API v5 is deprecated. Consider migrating to the latest version of the Twitch API, which is supported in Lumberyard by the newer Twitch API Gem (p. 1207).

For more information on integrating the Twitch v5 Gem with your Lumberyard project, see Engaging Broadcasters and Viewers on Twitch (p. 2095).

Enable the Twitch v5 Gem

To enable the Twitch v5 Gem, do the following:

1. Use Project Configurator (p. 44) to add the Twitch v5 Gem to your project. The Twitch v5 Gem requires the following Gem as a dependency:

   - HttpRequestor

2. Configure your project. Use the following command.

   `lmbr_waf configure`

3. Build your project. Use the following command.

   `lmbr_waf build_win_x64_vs2019_profile -p all --progress`

For more information on Gems, see the Gems system documentation (p. 1064).

Twitch API Gem

The Twitch API Gem is a wrapper for the Twitch RESTful API. Install this Gem into your Lumberyard project to access Twitch features through the Twitch API. This Gem makes it simple to engage broadcasters and leverage existing Twitch functionality for your game.

The Twitch API Gem does not provide a means for fetching information required to authenticate with the Twitch API, such as User ID, Application/Client ID, or Bearer tokens. You must first provide this information manually.
information to the Gem through EBus calls. Information about the types of Bearer tokens and how they can be acquired are detailed in the Authentication section of the Twitch developer documentation.

For more information on integrating the Twitch API Gem with your Lumberyard project, see Engaging Broadcasters and Viewers on Twitch (p. 2095).

**Enable the Twitch API Gem**

To enable the Twitch API Gem, do the following:

1. Use Project Configurator (p. 44) to add the TwitchApi Gem to your project. The TwitchApi Gem requires the following Gem as a dependency:
   - HttpRequestor
2. Configure your project. Use the following command.
   ```
   lmb_waf configure
   ```
3. Build your project. Use the following command.
   ```
   lmb_waf build_win_x64_vs2019_profile -p all --progress
   ```

For more information on Gems, see the Gems system documentation (p. 1064).

**Twitch Chat API Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Twitch Chat API Gem provides a flexible framework to create customized game interactions between streamers and spectators on Twitch, the world's leading social video platform and community for gamers. By adding Twitch ChatPlay to your project, you can quickly and easily connect to Twitch IRC via WebSocket.

Twitch ChatPlay includes support for chat commands, polls, and surveys that can be triggered by Twitch viewers through the Twitch chat channel. For example, you can create a chat command `#cheer` that triggers celebration animations in your game.

**Gem Setup**

Use of the Twitch Chat API Gem in your project requires the following setup:

1. Add the Twitch Chat API Gem and the Websockets Gem to your project.
   
   The Twitch Chat API Gem has a dependency on the Websockets Gem.
2. Setup an account on Twitch that will be used by the game. You will need to provide your username or a username that you wish to represent you (such as a bot id, potentially with mod permissions in the channel, depending on your goals).
3. Get a Twitch OAuth token.
   
   **Important**
   Keep this token safe and do not share this publicly! This is essentially your password for connecting, and if you share it with others, they can spoof your account.
For more information about **Twitch IRC** and authentication, see the [Twitch documentation](https://www.twitch.tv/docs/IRC).

You are now ready to use the Gem. For details on how to implement Twitch ChatPlay in your game, see [Intro to ChatPlay (p. 2096)](https://docs.aws.amazon.com/lumberyard/latest/userguide/chatplay-intro.html) and [Using the Twitch Chat API Gem (p. 2097)](https://docs.aws.amazon.com/lumberyard/latest/userguide/chatplay-twitch.html).

### UiBasics Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) or visit the [AWS Game Tech blog](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) to learn more.

The UiBasics Gem is a collection of assets to be used as defaults with the Lumberyard UI Editor, including basic UI prefabs (image, text, button, and text input) and the textures that those prefabs require. For more information, see [Creating and Customizing Project User Interfaces (p. 2874)](https://docs.aws.amazon.com/lumberyard/latest/userguide/project-ui.html).

### User Login Default Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) or visit the [AWS Game Tech blog](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) to learn more.

The UserLoginDefault Gem provides a default user login implementation for all platforms, which is useful for testing and debugging.

### Video Playback Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) or visit the [AWS Game Tech blog](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) to learn more.

With the Video Playback gem, you can play videos in a game or level loading screen, or on textures in your Lumberyard level. When you enable the Video Playback gem, the Video Playback component is enabled. For instructions on how to prepare Lumberyard for video playback, and how to set up video playback in your level, see [Video Playback (p. 906)](https://docs.aws.amazon.com/lumberyard/latest/userguide/video-playback.html).

For more information on playing videos in load screens, see [Defining Game and Level Load Screens (p. 2910)](https://docs.aws.amazon.com/lumberyard/latest/userguide/load-screens.html).

### Video Playback Bink Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) or visit the [AWS Game Tech blog](https://aws-gametech-platform.s3.amazonaws.com/download/03DE.txt) to learn more.

With the Video Playback Bink gem, you can play videos using RAD Game Tools' Bink video codec. The videos are playable in a game or level loading screen, or on textures in your Lumberyard level. When you enable the Video Playback Bink gem, the Video Playback Bink component is enabled. For instructions on how to prepare Lumberyard for video playback, and how to set up video playback in your level, see [Video Playback Bink (p. 913)](https://docs.aws.amazon.com/lumberyard/latest/userguide/video-playback-bink.html).
For more information on playing videos in load screens, see Defining Game and Level Load Screens (p. 2910).

**Important**
A Bink license is needed to gain access to this gem. Please contact RAD Game Tools for information on how to license Bink.

## Virtual Gamepad Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Virtual Gamepad gem to provide your game's UI with touch screen capabilities on mobile devices. After you enable the Virtual Gamepad gem, you can add the virtual gamepad components to your game's UI in the UI Editor.

**Topics**
- Configuring Virtual Gamepad Behavior (p. 1211)
- Virtual Gamepad Component Properties (p. 1213)
- Displaying the Virtual Gamepad at Runtime (p. 1214)

The Virtual Gamepad gem includes a sample UI canvas that you can either customize for your game or use as an example for a new UI canvas. To view this canvas in the UI Editor (p. 2874), open Gems\VirtualGamepad\Assets\UI\Canvases\VirtualGamepad\VirtualGamepad.uicanvas.

There is no limit for the number of active UI canvases that can contain virtual gamepad components. This means you can create many virtual gamepad canvases to display for the appropriate scenarios, or even at the same time. For example, you can show a different UI canvas on each half of your virtual gamepad.

**Example**

To experiment with the Virtual Gamepad gem, run the Advanced_RinLocomotion Sample (p. 151) level from the Samples Project (p. 146) on a device with touch screen support.
You can customize the virtual gamepad's appearance and layout by creating or modifying an existing UI canvas. For more information on working with UI Canvases, see Working with UI Canvases (p. 2888).

**Configuring Virtual Gamepad Behavior**

You can configure the virtual gamepad's behavior in the **UI Editor**.

**To configure virtual gamepad behavior**

1. In the **UI Editor**, add the UI components (p. 2923) **VirtualGamepadButton** and **VirtualGamepadThumbStick** to the UI canvas.
2. For each component, select an **Input Channel**.

For more information about input in Lumberyard, see [Input in Amazon Lumberyard](p. 1902).

To modify the list of available virtual gamepad input channels, use the Project Configurator to edit the virtual gamepad's system component. For more information, see [Configuring Advanced Settings](p. 49).
Virtual Gamepad Component Properties

The Virtual Gamepad gem features two components that you can use to customize input for your mobile games:

**VirtualGamepadButton**

**VirtualGamepadButton** has one property, **Input Channel**. Select the appropriate input.
VirtualGamepadThumbstick

**VirtualGamepadThumbstick** has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Channel</strong></td>
<td>The input channel that is updated when the user interacts with this control.</td>
</tr>
<tr>
<td><strong>Thumb Stick Image Centre</strong></td>
<td>The child element that is positioned at the center of the virtual thumbstick.</td>
</tr>
<tr>
<td><strong>Thumb Stick Image Radial</strong></td>
<td>The child element that is positioned beneath the user's finger when the virtual thumbstick is active. The image's position is always clamped to the radial edge of the virtual thumbstick center image.</td>
</tr>
<tr>
<td><strong>Centre When Pressed</strong></td>
<td>If selected, centers the virtual thumbstick when it is pressed.</td>
</tr>
<tr>
<td><strong>Adjust Position While Pressed</strong></td>
<td>If selected, adjusts the position of the virtual thumbstick when it is active. Tracks the player's finger when it moves outside the thumbstick radius.</td>
</tr>
</tbody>
</table>

**Displaying the Virtual Gamepad at Runtime**

You can enable the virtual gamepad for runtime by creating instructions that loads the virtual gamepad UI canvas. You can do this with C++, Lua, or Script Canvas. For more information, see *UICanvasManager (p. 3042)* and *UICanvasComponent (p. 3034)*.

The following example Lua script displays a virtual gamepad UI canvas if touch screen support is detected.

```lua
local touchDevice =
    InputDeviceRequestBus.Event.GetInputDevice(InputDeviceId(InputDeviceTouch.name))
if (touchDevice and touchDevice:IsSupported()) then
    print("Virtual gamepad UI canvas displayed")
end
```
The following example Lua script checks if a physical gamepad is connected. If found, the Lua script disables the virtual gamepad.

```lua
local gamepadDevice = InputDeviceRequestBus.Event.GetInputDevice(InputDeviceId(InputDeviceGamepad.name))
if (gamepadDevice and gamepadDevice:IsConnected()) then
  UiCanvasBus.Event.SetEnabled(self.virtualGamepadCanvasId, false);
end
```

The following Lua script enables the virtual gamepad when a physical gamepad is disconnected and disables the virtual gamepad when a physical gamepad is connected.

You can find a working example of this Lua script in the `lumberyard_version\dev\SamplesProject\AnimationSamples\Advanced_RinLocomotion\Scripts\Advanced_RinLocomotion.lua` file.

```lua
function Example:OnActivate()
  self.inputDeviceNotificationBus = InputDeviceNotificationBus.Connect(self);
end

function Example:OnInputDeviceConnectedEvent(inputDevice)
  if (inputDevice.deviceName == InputDeviceGamepad.name) then
    UiCanvasBus.Event.SetEnabled(self.virtualGamepadCanvasId, false);
  end
end

function Example:OnInputDeviceDisconnectedEvent(inputDevice)
  if (inputDevice.deviceName == InputDeviceGamepad.name) then
    UiCanvasBus.Event.SetEnabled(self.virtualGamepadCanvasId, true);
  end
end

function Example:OnDeactivate()
  if (self.inputDeviceNotificationBus) then
    self.inputDeviceNotificationBus:Disconnect();
  end
end
```

**Virtual Reality Gems**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Oculus and OpenVR Virtual Reality gems support a variety of Virtual Reality head-mounted displays (HMDs).

For more information about how to use and configure the Virtual Reality Gems, see Configuring your Project for Virtual Reality (p. 3249).

**Visibility**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can enable the Visibility gem for your game project so that you can add the VisArea (p. 918) component to an entity. You can use the component to create a visibility area (vis area) that helps manage visibility and culling in a scene.

Websockets Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Websockets Gem provides an easy way to create WebSocket clients for your Lumberyard games and projects. With this gem, you can connect to WebSocket servers and take advantage of the full-duplex communication that they provide.

This gem is designed to work with any platform-specific library that's needed. Each platform is divided out and separated, allowing libraries and features to be added quickly and easily, or to exchange existing or outdated tools with new ones. On Windows and Linux, the gem uses a well-known WebSocket library, WebSocket++, which is included in the gem.

The gem uses the WebSocket protocol to create clients that connect to existing WebSocket servers. Using the gem bus system, you can create multiple WebSocket connections, and then send and receive messages.

Websocket Requests Bus

The Websocket Request bus provides the primary interface for working with the gem. The follow functions are available:

- **CreateClient** – Provides the unique instance connection that will be used with the rest of the buses. It takes three arguments:
  1. The WebSocket server address.
  2. A received message handler function (OnMessage function).
  3. A type of connection. If no type is defined, it defaults to Secure.
- **IsConnectionOpen** – Used to check if the connection is open. Since WebSockets connections are always open, there is no need to verify if the connection has timed out, or to wait for a ping. Connections can be verified immediately.
- **SendWebsocketMessage** – Sends strings to the targeted server.
- **SendWebsocketMessageBinary** – Sends a binary stream to the targeted server.
- **CloseWebsocket** – The expectation for WebSocket connections is that the developer will close the WebSocket before the program is finished, allowing for a clean exit. This system will handle this automatically if not performed, but it is recommended, especially if there is a case where multiple connections are being setup simultaneously, to avoid having sockets open unnecessarily.
Tip
For an example of the usage of these functions, see the source for the Twitch Chat gem, which can be found in dev\Gems\TwitchChatPlay\.

IWebsocketClient
The IWebsocketClient class is an interface that sits at the base of the entire Websockets Gem. It provides a direct interface to the individual connections to WebSockets for users who wish to avoid the bus system in cases where speed is paramount. It provides all of the same functionality as the bus system once the connection has been created, with none of the overhead.

SecureWebsocketClient vs WebsocketClient
The Websockets Gem provides a method to create both secure and normal WebSocket clients, with the default being secure WebSockets connections. Secure WebSocket connections provide OpenSSL (Transport Layer Security) for protecting client-server connections using the standard included in Lumberyard.

OnMessage Function
Every WebSocket connection needs a message handler function to receive incoming messages from the server. The OnMessage function interface provides a simple method for you to provide this handler function. By providing a function pointer or lambda with the signature void (AZ::string_view), you can have your own custom functions handle responses, as shown in the following example:

```cpp
Websockets::OnMessage messageFunc = [this](const AZStd::string_view message)
{
    HandleResponses(message);
};
```

Known Issues
The following are known issues to be aware of when using this gem:

1. Amazon GameLift and the Websockets Gem are not currently compatible. Both GameLift and the gem use Websocket++, but unfortunately, the GameLift version is much older. In monolithic (release) builds, this creates linker issues that will cause a crash when both are active.

    Until a long-term fix is provided, you can make a simple change to the gem to fix this. Modify the WebSocket++ version within the gem to work with a new namespace. For example, change the websocketpp namespace to websocketppaz.

White Box Gem

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The White Box Gem provides two tools implemented as components; the White Box component, and the White Box Collision component. You can use the White Box component to quickly sketch proxy geometry for your entities. Use the White Box Collision component with the White Box component to provide a collision surface for PhysX simulations.
What is white boxing?

White boxing is a rapid, iterative design process. Simple modeling tools are used to create proxy geometry that defines the shape and dimension of 3D objects and environments. The proxy geometry created in this process has a very low polygon count, but reasonably approximates the size and shape of the final assets. White box entities have a basic material, and simple collision and physics materials if required. The entities created in the white box process can be handed off to artists to be developed into polished production entities.

In Lumberyard, you create entities with the White Box component, using its edit functionality to rapidly sketch 3D assets to serve as building blocks for a level. Because you create white box meshes in Lumberyard Editor with the component entity system, the entities can also contain functionality from other components such as scripts. You can use a rapid build and test iteration cycle to create refined and functional entities before committing to developing production-ready art assets. The resulting white box geometry and entities are used as templates for production assets. This development process is fast, highly iterative, and cost-effective.

Enable the White Box Gem

To make the White Box and White Box Collision components available in Lumberyard, you must build and configure your project with the White Box Gem enabled.

To enable the White Box Gem

1. Use Project Configurator to add the White Box Gem to your project.

   Note
   The White Box Gem has no dependencies. However, if you intend to use the White Box Collision component, you should also enable the PhysX Gem.

2. Configure your project. Use the following command.

   lmbr_waf configure

3. Build your project. Use the following command.

   lmbr_waf build_win_x64_vs2019_profile -p all --progress

For more information on Gems, see the Gems documentation (p. 1064).

White Box component information

For information on the White Box component, see White Box component (p. 946).

For information on the White Box Collision component, see White Box Collider component (p. 960).
Create levels in Lumberyard

A level, also known as world or map, represents the space or area available to the player during the course of completing a discrete game objective. Most games consist of multiple levels through which a player can advance to or move through, although usually only a single level is loaded at a time. Each level can be grouped into multiple layers, which you use to logically group types of objects.

The environment includes lighting, terrain, bodies of water, vegetation, sky, and weather effects.

Topics
- Creating a New Level (p. 1219)
- Creating Terrain (p. 1220)
- Blending Environment Probes (p. 1252)
- Working with Database Libraries (p. 1264)
- Adding Sky Effects (p. 1265)
- Adding Weather Effects (p. 1290)
- Adding Vegetation (p. 1298)

Creating a New Level

The first step in creating a game world is to create a level.

To create a new level
1. In Lumberyard Editor, click File, New Level or press Control + N.
2. In the **New Level** window, enter a file name and select a directory location for the file. Choose **OK**.

3. In the **Generate Terrain Texture** window, for **Texture Dimensions**, select texture dimensions to determine texture tile resolution. Choose **OK**.

## Creating Terrain

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can add realistic elements such as mountains, valleys, lakes, rivers, and roads to your terrain for your environment levels.

One of the primary tools used to first create a terrain is the Terrain Editor, as the following shows:

---

**Topics**

- Creating High Quality Terrain (p. 1221)
- Importing and Exporting Terrain Data (p. 1222)
- Using the Terrain Heightmap (p. 1226)
- Using Terrain Texture Layers (p. 1232)
- Creating Landforms and Topography (p. 1237)
- Creating Bodies of water (p. 1242)
Creating High Quality Terrain

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you create a level, you can refine your terrain to use 2k height and 16k color.

To refine your terrain to a 2k world with 16k color

1. In your Lumberyard Editor level, choose Game, Terrain, Refine Terrain Texture Tiles.
2. In the dialog box, click Yes and in the next dialog, click OK.
3. Save your level.
4. Repeat steps 1 through 3.
5. Choose Game, Terrain, Export/Import Megaterrain Texture.
6. In the Export/Import Megaterrain Texture tool, click and drag to select all of the tiles.
7. Click Change tile resolution, select 2048x2048, and then click OK.

Your result should look like the following. You can now import and export high quality textures.
Tip
You can verify that your texture is in 16k color. Export and then open the texture file in a DCC such as Photoshop.

Importing and Exporting Terrain Data

You can import and export terrain data such as color maps, terrain blocks, and splat maps.

Topics
- Importing a Megaterrain Texture (p. 1222)
- Importing and Exporting Terrain Blocks (p. 1222)
- Importing Splat Maps (p. 1223)
- Terrain FAQs (p. 1224)

Importing a Megaterrain Texture

A megaterrain texture, also known as a color map, is a texture that covers part of or the entire terrain. This texture is visible from a distance and presents sweeping vistas. As the camera moves closer to the terrain, the megaterrain texture is replaced by more detailed terrain textures.

Although you can import data for individual tiles, a more common workflow is to use a single megaterrain texture for the entire terrain.

Note
Lumberyard doesn't match or fix tile edges. Ensure that your tile data is authored so that adjacent tile edges match. Otherwise, your level may render with visible seams. Lumberyard supports the BMP image file type only for megatexture terrains.

To import a megaterrain texture

1. In Lumberyard Editor, choose Tools, Terrain Editor.
2. In the Terrain Editor, choose Tools, Export/Import Megaterrain Texture.
3. Click and drag to select the tiles that you want.
4. Click Import, select your megaterrain texture file, and then click Open.

Importing and Exporting Terrain Blocks
You can import and export terrain areas or blocks. When importing or exporting, you should also import or export the associated terrain texture layers.

**To import a terrain block and texture layers**

1. In Lumberyard Editor, click **Game, Terrain, Import Terrain Block** and select a `.trb` file to import.
2. Click **Terrain, Terrain Texture Layers**.
3. Click **File, Import Layers** and select terrain texture files to import.

**To export a terrain block and texture layers**

1. In Lumberyard Editor, click **Game, Terrain, Export Terrain Block**, and select a `.trb` file to export.
2. Click **Terrain, Terrain Texture Layers**.
3. Click **File, Export Layers**, and select terrain texture files to export.

**Importing Splat Maps**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Splat maps are 8-bit monochrome bitmap `.bmp` files that contain weight information for each vertex in a terrain map. Splat maps are generated using a DCC tool such as World Machine's Splat Converter.
All splat map operations in Lumberyard are done using the Terrain Texture Layers editor.

**To import splat maps**

1. In Lumberyard Editor, choose **Tools, Other, Terrain Texture Layers** (p. 1232).
2. In the Terrain Texture Layers editor, under **Layer Tasks**, assign each splat map to a texture layer by clicking a layer and then clicking **Assign Splat Map**.
3. When prompted, select a .bmp file. You don't need to assign a splat map path to a layer, but you can't assign more than one path either.
4. Under **Layer Tasks**, click **Import Splat Maps**. This clears the current weight map for the terrain and then rebuilds it using the selected splat maps.
5. In Lumberyard Editor, choose **Game, Terrain, Generate Terrain Texture**.

**Note**
You cannot apply masking to an imported splat map.

**Terrain FAQs**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/serverless) or visit the [AWS Game Tech blog](https://aws.amazon.com/blogs/gaming) to learn more.

See the following troubleshooting tips to resolve issues for your Lumberyard terrains.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Workaround or Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can I center the terrain in world space?</td>
<td>Terrain in Lumberyard is generated from a texture. Texture coordinates (U,V) originate in the lower left corner of the texture. Because terrain is based on a terrain texture, the Legacy Terrain level component is placed with its lower left corner at X: 0.0 and Y: 0.0 in world space. The terrain is placed at Z: 32.0, with an Infinite Ocean below it at Z: 16.0 in world space. This terrain placement has the useful byproduct of ensuring every entity in a level is placed at a positive X and Y coordinate, and that you can create valleys and bodies of water within the terrain.</td>
</tr>
<tr>
<td>Issue</td>
<td>Workaround or Fix</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Lumberyard terrain can't be moved. If your project requires the terrain to be centered at zero in world space, you can remove the Legacy Terrain level component, and remove the Infinite Ocean in the Terrain Editor. You can import a terrain mesh created in a third-party application as you would any other .fbx mesh asset, and place the terrain in the level as an entity at any world space position.</td>
<td></td>
</tr>
<tr>
<td>I expected to create a 2k world, but the Terrain Editor shows tiles that are sized 512 in a 2x2 grid. What does this mean?</td>
<td>512 describes the number of megatexels per tile and is the default resolution for new levels in Lumberyard. With a 2x2 grid, you have a total of four tiles. If you add up all of the megatexels from all of the tiles, it adds up to 2k (2048 megatexels). To increase your tiles' resolution, see Creating High Quality Terrain (p. 1221).</td>
</tr>
<tr>
<td>What does the resolution (512x512, 1024x1024, 2048x2048) mean in terms of the 2k height and the 16k size of the megaterrain?</td>
<td>The heightmap is independent of the megaterrain texture.</td>
</tr>
<tr>
<td>The resolution displayed in the Export/Import Megaterrain Texture dialog is the color megatexel resolution for that tile.</td>
<td></td>
</tr>
<tr>
<td>What is the optimal resolution for imported splat maps?</td>
<td>You should match the heightmap resolution for your splat map resolutions.</td>
</tr>
<tr>
<td>Why does the visual quality get blurry after generating terrain textures? It seems that sectors start fetching mips at various resolutions, making it difficult to determine what it looks like.</td>
<td>Lumberyard uses a texture pool for terrain textures. Its default is set to retain 64 textures at any given time, with each texture measuring 256x256. Consider that a 16k x 16k texture actually contains 5461 individual textures. By default, Lumberyard doesn't keep enough textures in resident memory at any given time to present a clear image. You can increase the texture pool size using the e_TerrainTextureStreamingPoolItemsNum console variable. The maximum value is 4096. You can set this console variable in the level.cfg or project.cfg file. You must set the console variable in one of these files because the value must be present when the system loads. Changes made during runtime don't affect this console variable. For more information, see Using the Console Window (p. 210).</td>
</tr>
</tbody>
</table>

Version 1.28
1225
### Issue

How are terrain layer materials applied to the terrain? What does the **detail repeat** property do?

- For example, if a dirt map is 2m x 2m, what is the proper detail repeat value to use in the material?
- Are UVs 0..1 per meter/quad?
- How does detail repeat values relate to overall terrain?

<table>
<thead>
<tr>
<th>Issue</th>
<th>Workaround or Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>How are terrain layer materials applied to the terrain? What does the <strong>detail repeat</strong> property do?</td>
<td>If you author a terrain detail material as a 2m patch, then set the TileU and TileV in the material to 0.5. The shader calculates UV coordinates from world space. Data is passed into the terrain shader that tells it the x-y offset and scale for the texture UVs. The shader then uses that to transform the world space coordinate into a UV coordinate.</td>
</tr>
</tbody>
</table>

### Creating a Terrain Heightmap

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The heightmap is the base of the terrain in your level. You have three options for obtaining a terrain heightmap:

- Create a new heightmap using the Terrain Editor
- Create a new heightmap using a third-party terrain-building tool
- Importing an existing heightmap

#### Topics

- Creating a Terrain Heightmap (p. 1226)
- Setting Heightmap Properties (p. 1228)
- Importing a Terrain Heightmap (p. 1230)
- Exporting a Terrain Heightmap (p. 1231)
- Resizing a Terrain Heightmap (p. 1231)
- Rotating a Terrain Heightmap (p. 1231)

### Creating a Terrain Heightmap

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The first step in creating the heightmap using Lumberyard Editor is to specify the resolution and grid spacing, both of which define the terrain size. Terrain size is determined by multiplying heightmap resolution by meters per texel. This value should not exceed 4096 x 4096 kilometers.

Meters per texel is the distance in meters between two vertices on the grid. So a value of two means there is a grid point every two meters. You can use larger values to create a larger terrain, but it is less detailed for the same heightmap resolution.

The following images show a terrain heightmap and the corresponding generated terrain.
Example Terrain Heightmap

![Example Terrain Heightmap](image1.jpg)

Example Generated Terrain

![Example Generated Terrain](image2.jpg)
If you want to create the highest quality terrain heightmap for a 2K world, use the following settings:

- 2K heightmap
- 2K splatmaps

As the splat weighting is applied and stored in the terrain mesh, the splatmaps should match the terrain heightmap resolution.

- 4K – 16K megaterrain color map

To create a heightmap using the Terrain Editor

1. In Lumberyard Editor, click File, New.
2. In New Level, enter a file name and directory location for the heightmap file.
3. Select the desired Heightmap Resolution and Meters per Texel values and click OK.
4. In the editor menu, click Game, Terrain, Edit Terrain.
5. In Terrain Editor, click Tools, Generate Terrain.
6. In Generation, adjust the following parameter values as needed.

   **Feature Size**
   
   Determines the amount of land created.

   **Bumpiness/Noise (Fade)**
   
   Determines the degree of bumpiness or deformation of the surface.

   **Slope Detail (Passes)**
   
   Determines the number of times that effect is applied.

   **Seed (Random Base)**
   
   Determines the degree of random variation for the heightmap.

   **Slope Smoothing (Blur Passes)**
   
   Sets the number of times that smoothing is applied to the noise filter.

   **Cover (Exp. Substract)**

   **Sharpness (Exp. Base)**
   
   Determines the sharpness of the surface.

   **Sharpness (Freq. Step)**
   
   Determines the number of times that the sharpness filter is applied to the surface.

**Setting Heightmap Properties**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Terrain Editor to set various heightmap properties and parameters that affect the shape of the terrain profile.

Version 1.28

1228
To set heightmap properties

1. In Lumberyard Editor, click **Game, Terrain, Edit Terrain**.
2. In **Terrain Editor**, click **Modify**, and then click and adjust the various following properties and parameters:
   - **Make Isle**
     - Sinks the heightmap so that it is surrounded by ocean.
   - **Remove Ocean**
     - Sets the ocean level to –100000 meters.
   - **Set Ocean Height**
     - Sets the ocean level in meters.
   - **Set Terrain Max Height**
     - Sets the maximum height for the tallest mountain. (Default is 1024 meters).
   - **Set Unit Size**
     - Sets the meters per texel size of the heightmap.
   - **Flatten**
     - Flattens terrain to either a higher or lower point.
   - **Smooth**
     - Removes all hard edges from the heightmap.
   - **Smooth Slope**
     - Removes hard edges from steep areas of the heightmap.
   - **Smooth Beaches/Coast**
     - Removes hard edges from flat areas of the heightmap.
   - **Normalize**
     - Ensures the entire greyscale spectrum is used between the **Max Height** value and zero.
   - **Reduce Range (Light)**
     - Makes heightmap mountains smaller.
   - **Reduce Range (Heavy)**
     - Makes heightmap mountains small.
   - **Erase Terrain**
     - Deletes all heightmap data.
   - **Resize Terrain**
     - Resizes the terrain heightmap.
   - **Invert Terrain**
     - Inverts all greyscale data, changing black to white and vice versa.
Importing a Terrain Heightmap

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In order to support full, 32-bit heightmap resolution, use one of the following file formats when you import a heightmap file.

- `.asc`
- `.bt` (binary terrain) – Highest precision
- `.tif` (32-bit)

The following file formats are supported but not recommended. These file formats are imported in 16-bit or 8-bit resolution. We recommend using a minimum of 16-bit resolution for heightmaps, as 8-bit resolution can cause blocky-looking terrain.

- `.pgm` (16-bit)
- `.raw` (16-bit)
- `.r16` (16-bit)
- `.png` (8-bit)
- `.bmp` (8-bit)
- `.tif` (8-bit or 16-bit)

Lumberyard Editor supports importing only a single heightmap for the entire terrain. You can, however, import color or texture data to individual tiles within the terrain. For information about importing color maps, see Importing a Megaterrain Texture (p. 1222).

If you attempt to import a file with a different resolution than your existing heightmap, Lumberyard prompts you to do one of the following:

- Clip your imported file to remove any values that are outside the boundary of your existing heightmap.
- Resize your imported file to shrink or stretch to fit the heightmap. Resizing may introduce artifacts that can cause blocky-looking terrain.

**Note**
If your heightmap resolution is greater than your level's resolution, you should downsample your heightmap with an external graphics application before you import it into Lumberyard. If, for example, you attempt to import a 4k heightmap into a 2k level, Lumberyard prompts you to clip or resize your heightmap. This may be sufficient, but using an external graphics application to downsample your heightmap provides superior results.

To import a heightmap

1. In Lumberyard Editor, choose Game, Terrain, Edit Terrain.
2. In Terrain Editor, choose File, Import Heightmap.
Exporting a Terrain Heightmap

You can export a heightmap file that was created in the Terrain Editor to the following file formats:

- .asc (32-bit)
- .bt (32-bit)
- .tif (32-bit)
- .pgm (16-bit)
- .raw (16-bit)
- .r16 (16-bit)
- .bmp (8-bit)
- .png (8-bit)

To export a heightmap

1. In Lumberyard Editor, choose Game, Terrain, Edit Terrain.
2. In Terrain Editor, choose File, Export Heightmap and enter a file name and directory location.

Resizing a Terrain Heightmap

Resizing the terrain heightmap involves changing the resolution of your heightmap. Terrain size is determined by multiplying heightmap by meters per texel. When resizing, this value should not exceed 4096x4096 meters.

Meters per texel is the distance in meters between two vertices on the grid. So a value of two means there is a grid point every two meters. You can use larger values to create a larger terrain, but it is less detailed for the same heightmap resolution.

To resize a heightmap

1. In Lumberyard Editor, click Game, Terrain, Resize Terrain.
2. For Heightmap Resolution, select the desired value.
3. For Meters Per Texel, select the desired value.

Rotating a Terrain Heightmap

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
Rotating a terrain heightmap involves just a few simple steps.

**To rotate the heightmap**

1. In the Rollup Bar, on the Terrain tab, click Move Area.
2. Click Select Source.
3. At the bottom of Lumberyard Editor, enter the X and Y coordinates for the heightmap center, then click Lock Selection.
4. Click Select Target and repeat Step 3.
5. In Target Rotation, select a value in degrees.

**Using Terrain Texture Layers**

You can create terrain texture layers and paint them to enhance your environment level.

The primary tool for creating and managing terrain texture layers in your level is the Terrain Texture Layers editor.

For information about working with splat maps, see .
Creating Terrain

- Adding a Terrain Texture Layer (p. 1233)
- Applying a Texture Layer Material (p. 1233)
- Importing Terrain Texture Layers (p. 1233)
- Exporting Terrain Texture Layers (p. 1234)
- Painting Terrain Texture Layers (p. 1234)
- Changing Terrain Tile Resolution (p. 1236)

Adding a Terrain Texture Layer

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Create a new texture layer and change the terrain texture and material as desired.

To add a terrain texture layer

1. In Lumberyard Editor, click Tools, Other, Terrain Texture Layers.
2. In Terrain Texture Layers, under Layer Tasks, click Add Layer.
3. Double-click the NewLayer text and assign a unique name to it.
4. Click Materials\material_terrain_default to open Material Editor.
5. In the tree, click Materials\material_terrain_default and adjust material and other settings as needed.

Applying a Texture Layer Material

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

All terrain texture layer materials use the Terrain.Layer Shader (p. 1710). All terrain materials should be "high-passed" in your DCC tool in order for them to work correctly with this shader.

To apply or edit a material for a texture layer

1. In Lumberyard Editor, click Tools, Other, Terrain Texture Layers.
2. In Terrain Texture Layers, double-click the layer you want to apply or edit a material for.
3. In Material Editor, expand the tree and select your asset.
4. Change settings and shader parameters as needed.
6. Close Material Editor.

Importing Terrain Texture Layers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

By importing a saved layer, all materials, textures, and shader settings can be quickly applied to your level.
To import a terrain texture layer
1. In Lumberyard Editor, click Tools, Other, Terrain Texture Layers.
2. Click File, Import Layers.
3. Select the layer (.lay) file for import, then click Open.

Exporting Terrain Texture Layers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can export your terrain texture layer, so that you can then reuse it in multiple levels.

To export a terrain texture layer
1. In Lumberyard Editor, choose Tools, Other, Terrain Texture Layers.
2. Click File, Export Layers.
3. Enter a file name and select a directory path for the exported file and then click Save.

Painting Terrain Texture Layers

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard uses two components for painting terrain texture layers:

- The first is a low-resolution texture with color information. This texture is visible from a distance and provides underlying color information for the base terrain texture. This texture should be less than 512 x 512 pixels in size.
- The second is a high-resolution material. This material is visible at close distances and can have several texture maps like diffuse, bump, and specular. The diffuse map should be set to white (255).

The distance at which low-resolution textures are replaced with those of a higher resolution is defined by the DetailLayersViewDistRatio parameter. To access this parameter, in the Rollup Bar, on the Terrain tab, click Environment and adjust the value as needed.

To paint a terrain texture layer
1. In Lumberyard Editor, choose Game, Terrain, Paint Layers.
2. In the Rollup Bar, on the Terrain tab, click Layer Painter.
3. Adjust the terrain brush settings as needed.
Radius

Specifies the size of the brush. Use the slider to adjust the size, or use the following keyboard shortcuts: [ to increase the brush radius size or ] to decrease the brush radius size.

Color Opacity

Specifies the strength of the brush when applying the layer color. The brush is a spray brush that uses the color opacity value to determine the opacity at the center of the brush. The opacity decreases to transparent at the edge of the brush. Lower values create a more translucent brush, and higher values create a more opaque brush. Set the value to 0 to disable the color opacity. You can use the slider to adjust the opacity level, or you can use the following keyboard shortcuts:

- **Shift+[** decreases the opacity.
- **Shift+]** increases the opacity.

You can also use the following keyboard shortcuts to adjust both parameters simultaneously with the same value:

- **Shift+Ctrl+[** decreases both color opacity and detail intensity.
- **Shift+Ctrl+]** increases both color opacity and detail intensity.

Detail Intensity

Specifies the strength of the brush when applying the detail texture to the layer. Like color opacity, the detail intensity value determines the intensity (opacity) at the center of the brush. The opacity decreases to transparent at the edge of the brush. Lower values create a more translucent (less intense) texture, and higher values create a more opaque texture. Set the value to 0 to disable detail intensity. You can use the slider to adjust the intensity level, or you can use the following keyboard shortcuts:

- **Ctrl+[** decreases the intensity.
- **Ctrl+]** increases the intensity.

You can also use the following keyboard shortcuts to adjust both parameters simultaneously with the same value:

- **Shift+Ctrl+[** decreases both color opacity and detail intensity.
- **Shift+Ctrl+]** increases both color opacity and detail intensity.

Mask by Layer Altitude and Slope

Sets the material to paint only between the layer Altitude and Slope parameters defined below.

Mask by

Selects a layer to prevent it from being painted over.

4. Adjust the layer brush settings as needed.

Brightness

Modifies the brightness of the material base color. Click the Color box to open up the color selector and alter the base color of your material. Click Save Layer when done.

Altitude

Sets a minimum and maximum altitude mask for painting; the brush applies only within these boundaries.

Slope (degrees)

Sets a minimum and maximum slope mask for painting; the brush applies only within these boundaries.
Changing Terrain Tile Resolution

A terrain layer can be divided into multiple tiles, each of which can be painted with a resolution between 64x64 and 2048x2048 kilometers. The higher the resolution, the softer the transition between terrain texture layers.

If you know a player spends a lot of time in specific areas of the level and thus have more opportunity to view the terrain, you can save resources by increasing the resolution in just those areas. Follow this two-step process:

You first subdivide the texture layer, then change the individual tile resolution, as follows:

**To subdivide the terrain texture layer**

1. In Lumberyard Editor, click **Tools, Other, Terrain Texture Layers**.
2. Click **File, Refine Terrain Texture Tiles**. The layer is now split into 2x2 (4) tiles.
3. Repeat step 2. The layer is now divided into 4x4 (16) tiles.
4. Repeat only as needed as there is no way to go back and reduce the number of tiles.

**To change terrain tile resolution**

1. In Lumberyard Editor, click **Game, Terrain, Export/Import Megaterrain Texture**.
2. In **Terrain Texture**, click a tile whose resolution you want to change. Then click **Change tile resolution**.
3. Choose a new resolution and then click OK. Click Close.
Creating Terrain

Creating Landforms and Topography

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add realistic mountains, hills, valleys, and other landforms to your terrain in your environment level. The primary method for creating interesting terrain features and landforms involves the following brushes:

- **Rise/Lower brush** – Increases and decreases the local terrain height to quickly create hills, valleys, and river beds, for example.
- **Flatten brush** – Flattens the terrain at a specified height and diameter. Use the **Pick Height** feature to select a height from which to begin flattening.
- **Smooth brush** – Smooths over sharp gradients in the terrain.
- **Holes brush** – Used to make holes in the terrain for creating areas beneath or inside the terrain such as caves.

You can also use the Terrain Editor to modify the terrain heightmap, although this method is not as accurate and does not give you the control you get from working directly in the viewport in Lumberyard Editor. For more information, see Setting Heightmap Properties (p. 1228).

**Topics**

- Using the Rise/Lower Brush (p. 1237)
- Using the Smooth Brush (p. 1238)
- Using the Flatten Brush (p. 1238)
- Using the Holes Brush (p. 1239)
- Terrain Brush Parameters (p. 1239)
- Creating Roads (p. 1240)

**Using the Rise/Lower Brush**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

The Rise/Lower brush is perhaps the most versatile of all the terrain brushes and is often the first used. With it you can create many macroterrain landforms and features such as mountains, hills, cliffs, valleys, and riverbeds, for example. After using this brush, see Using the Flatten Brush (p. 1238) and Using the Smooth Brush (p. 1238) to learn how to control the shape and overall visual look.

**To use the Rise/Lower brush**

1. In the Rollup Bar, on the Terrain tab, click Modify, Rise/Lower.
2. Adjust the Height slider to the desired height:
Creating Terrain

- Use positive values for landforms that rise above the base level.
- Use negative values for valleys and other landforms that sink below the base level.

3. Adjust the **Outside Radius** and **Inner Radius** sliders (and the difference between the values) to control the steepness of the terrain.

4. In the level, drag the mouse around to achieve the desired effect.

5. Under **Modify Terrain**, adjust **Brush Settings** and **Noise Settings** parameters as needed. See Terrain Brush Parameters (p. 1239) for more information.

6. When done, click **Terrain**, **Modify** or press **Esc**.

### Using the Smooth Brush

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Smooth brush softens sharp gradients in the terrain, such as the sides of mountains, cliffs and lake beds for example. This brush averages out the height of the terrain based on nearby terrain areas to provide a smoother surface.

**To use the smooth brush**

1. In the **Rollup Bar**, on the **Terrain** tab, click **Modify**, **Smooth**.
2. In the level, drag the mouse to create the smoothing effect.
3. Under **Modify Terrain**, adjust **Brush Settings** and **Noise Settings** parameters as needed. See Terrain Brush Parameters (p. 1239) for more information.
4. When done, click **Terrain**, **Modify** or press **Esc**.

### Using the Flatten Brush

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Flatten brush makes any piece of terrain completely flat at a height that you define. This is useful for creating a variety of features such as plateaus, mesas, and buttes as well as creating flat spots wherever needed.

**To use the Flatten brush**

1. In the **Rollup Bar**, on the **Terrain** tab, click **Modify**, **Flatten**.
2. In the level, drag the mouse to create a flat spot. The terrain is flattened at the selected **Height** and **Diameter** brush settings.

3. Under **Modify Terrain**, adjust **Brush Settings** and **Noise Settings** parameters as needed. See **Terrain Brush Parameters (p. 1239)** for more information.

### Using the Holes Brush

This topic references tools and features that are **legacy**. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the **Lumberyard Legacy Reference**.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Holes brush makes a geometrical hole in both the terrain layer as well as the visual mesh. It is useful for creating craters, sinkholes, caves, and other areas beneath or inside the terrain. The resulting holes can be filled with various objects such as rocks or vegetation.

**To use the Holes brush**

1. In the **Rollup Bar**, on the **Terrain** tab, click **Holes**.
2. Adjust the **Brush Radius** slider to adjust the size of the hole.
3. Click **Make Hole** to create a hole.
4. In the level, click to place the hole. By default you can see the ocean showing through.
5. To remove a hole, click **Remove Hole**. You are limited to removing one terrain unit adjacent to the existing terrain.

### Terrain Brush Parameters

A number of settings apply to multiple terrain brushes. Use the following parameters to adjust the rise/lower, smooth, and flatten brushes.

**Outside Radius**

The outer edge of the area of the terrain brush effect.

**Sync Radius for All Types**

Select to set the same outer radius value across the flatten, smooth, and rise/lower brushes.

**Inside Radius**

The inner edge of the area of the terrain brush effect. Within this radius the effect of the brush is at its maximum.

**Hardness**

Controls the shape of the fall-off curve between the inner and outer radius of the brush.
Creating Terrain

**Height**
For the rise/lower and flatten brushes, the incremental amount the terrain is be raised/lowered or flattened with each click in the terrain level.

**Enable Noise**
Select to add random terrain variances to the brush.

**Scale**
Controls the strength of the noise effect.

**Frequency**
How often the noise effect is applied.

**Reposition Objects**
Select to realign objects with the modified terrain. Objects remain on top.

**Reposition Vegetation**
Select to realign vegetation with the modified terrain. Vegetation remains on top.

---

**Creating Roads**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add realistic roads to your terrain in your environment level.

For information on the road entity, see Road Entity.

**Topics**
- Creating the Road Entity (p. 1240)
- Applying a Road Material (p. 1241)
- Adjusting Road Spline Geometry (p. 1241)
- Splitting and Merging Roads (p. 1242)

---

**Creating the Road Entity**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create and place roads using the Road entity as follows.

When performing this procedure, you may notice that parts of the road disappear into the terrain. The **Align Height Map** step resolves this by stretching the terrain height to match the path of the road based
on its shape and on **BorderWidth** parameter. For information on **BorderWidth** and related settings, see **Road Entity**.

**To create and place the Road entity**

1. In the **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. In your level, start at the beginning of the road and click to place a series of points that define the road’s path.
3. When complete, double-click where you want the road to end.
4. In the **Rollup Bar**, under **Road Parameters**, click **Align Height Map** to adjust the terrain height to match the path of the road.

**Applying a Road Material**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After the Road entity has been placed, you can apply a material to the road.

**To apply a material to a road**

1. In the **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. Click **<No Custom Material>** to open the Material Editor.
3. In the Material Editor, expand the tree and select your asset.
4. Modify material settings and shader parameters as needed.
5. When finished, click **Assign Item to Selected Objects**, and close the Material Editor.

**Adjusting Road Spline Geometry**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can make precise changes to the geometry of a road by adjusting the spline points and parameters.

**To adjust road spline parameters**

1. In the **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. Under **Spline Parameters**, click **Edit**, and do any of the following for the road in your level:
Creating Terrain

- To move a point, drag it.
- To add a new point, hold down Ctrl while you click on the spline at the desired location.
- To delete a point, double-click it.
- To change the angle at a point, select it and adjust the Angle value.
- To change the width at a point, select it, clear the Default width check box, and adjust the Width value.

Splitting and Merging Roads

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can split a road apart or merge two roads together.

To split or merge roads

1. In the Rollup Bar, on the Objects tab, click Misc, Road.
2. Under Spline Parameters, click Edit, and do the following for the road in your level:
   - To split a road apart, select the desired point and click Split.
   - To merge two roads together, select the end point of one road and the start point of another road. Then click Merge.

Creating Bodies of water

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create realistic-looking ocean, lakes, rivers, waterfalls, and pools with waves and ripples. Players and objects interacting with water surfaces also generate waves and ripples. Water gets its appearance from reflections on the surface and the interaction of light with particles suspended underneath the surface. You need both to achieve an authentic look.

Lumberyard offers three different shaders for rendering bodies of water:

- Water Shader (p. 1715) – For oceans only
- WaterVolume Shader (p. 1719) – For lakes, rivers, ponds and all other water volumes
- Waterfall Shader (p. 1717) – For waterfalls only

Lumberyard also supports caustics. Caustics are optical properties caused by light refracting through a volume of water, creating light and dark patterns at the bottom. Realistic caustic effects also include water ripples generated from players and other objects interacting with the water surface.
Note
To make caustics visible, you must place water volumes at a height of 1 or greater in your level.

Topics
- Preparing the Terrain (p. 1243)
- Setting Ocean Parameters (p. 1243)
- Creating Rivers (p. 1244)
- Adding Waterfalls (p. 1248)
- Adding Water Puddles (p. 1248)
- Adding Fog Beneath Water (p. 1249)
- Advanced Water Volume Parameters (p. 1250)

Preparing the Terrain

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For all water volumes such as lakes, ponds, and reservoirs, the terrain must first be lowered and sculpted to contain the body of water. To create the bottom and walls of your body of water, you need to consider the depth, shape, and edges of your landform geography.

For rivers, see Preparing the River Terrain (p. 1244).

To prepare the terrain for bodies of water

1. In the Rollup Bar, on the Terrain tab, click Modify, Rise/Lower.
2. Adjust the Outside Radius value as needed for the widest point of the water volume.
3. Adjust the Height value to a negative value for the depth of the water volume.
4. Adjust the other terrain brush settings as needed to fine-tune the look of the walls. See Terrain Brush Parameters (p. 1239) for more information.
5. In your level, drag to define the shape. Release the mouse button and repeat as needed.

Setting Ocean Parameters

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
When you create a new level, Lumberyard creates an ocean by default, complete with waves and reflections. The ocean uses the Water Shader (p. 1715). You can change the ocean's various properties and effects.

To set ocean parameters

1. In the Rollup Bar, on the Terrain tab, click Environment.
2. Under Ocean, adjust the following parameter values:
   - Material – Click the (...) icon to access Material Editor and select your asset.
   - Caustic depth – Set the depth to which caustic effects are visible.
   - Caustic intensity – Scale the intensity of the caustics for the water surface normals.
   - Caustic tiling – Scale the caustic tiling applied to the water surface normals. You can scale caustics independently of the surface material in cases of strong tiled normals or vice-versa.

Creating Rivers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add realistic rivers, complete with waterfalls, to your terrain in your environment level.

The following are best practices and guidelines to keep in mind when creating rivers.

- Rivers are 2D objects, which means rivers cannot be made to flow down steep inclines. However, to make a river flow down gentle inclines, you can rotate the river along the z-axis slightly (Z=0.5 to 1.0).
- To create rivers that appear to flow down steep inclines, create multiple rivers and connect them with waterfalls.
- The more points you place for the river geometry, the more control you have for direction and curvature.
- The wider the river, the further apart the points should be to avoid clipping at sharp corners.
- For more realism, paint the bottom of the river a different texture and add vegetation.
- For more realism, add particle effects.

For information on the river entity see River Entity.

Topics

- Preparing the River Terrain (p. 1244)
- Creating the River Entity (p. 1245)
- Applying a River Material (p. 1246)
- Adjusting River Spline Geometry (p. 1247)
- Splitting and Merging Rivers (p. 1247)

Preparing the River Terrain

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
Rivers need a riverbed and walls, which you implement as a deformation in the terrain. Use the rise/lower terrain brush for this effect.

To create a realistic-looking riverbed and walls, make sure that the walls of the river are above the starting (first) point of the river for the entire length of the river.

**To create the riverbed and walls**

1. In the Rollup Bar, on the Terrain tab, click Modify, Rise/Lower.
2. Adjust the Outside Radius slider as needed for the width of the riverbed.
3. Adjust the Height slider to a negative value for the depth of the riverbed.
4. Adjust the other terrain brush settings as needed to fine-tune the look of the riverbed. See Terrain Brush Parameters (p. 1239) for more information.
5. In your level, position the mouse at the start of river, and then drag to define the direction and course of the river. Release the mouse at the end of the river.

**Creating the River Entity**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

After you have prepared the riverbed, you next create and place the River entity.

When performing this procedure, you may notice that parts of the river disappear into the terrain. The Align Height Map step resolves this by stretching the terrain height to match the path of the river based on its shape and on BorderWidth parameter. For information on BorderWidth and related settings, see River Entity.
To create and place the River entity

1. In the Rollup Bar, on the Objects tab, click Misc, River.
2. In your level, starting at the beginning of the river bed, click to place a series of points that define the river’s path.
3. When complete, double-click at the end of the river bed.
4. In the Rollup Bar, under River Parameters, click Align Height Map to adjust the terrain height to match the path of the river.

Applying a River Material

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you place the river entity, you can apply a material to the river. Rivers use the WaterVolume Shader (p. 1719).

To apply a material to a river

1. In the Rollup Bar, on the Objects tab, click Misc, River.
Creating Terrain

2. Click `<No Custom Material>` to open Material Editor.
3. In the Material Editor, expand the tree and select your asset.
4. Modify material settings and WaterVolume Shader (p. 1719) parameters as needed.
5. When finished, click Assign Item to Selected Objects, and close Material Editor.

Adjusting River Spline Geometry

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can make precise changes to the geometry of a river. You simply adjust the spline points and parameters.

To adjust river spline parameters

1. In the Rollup Bar, on the Objects tab, click Misc, River.
2. Under Spline Parameters, click Edit, and do any of the following for the river in your level:
   - To move a point, drag it.
   - To add a new point, hold down Ctrl while clicking the spline at the desired location.
   - To delete a point, double-click it.
   - To change the angle at a point, select it and adjust the Angle value.
   - To change the width at a point, select it, clear the Default width check box, and adjust the Width value.

Tip
You can also change the positions of any spline point. Just select a point and use the X, Y, Z, and XY axis-lock buttons located at the top of Lumberyard Editor.

Splitting and Merging Rivers

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can split a river apart and merge two rivers together.

To split or merge rivers

1. In the Rollup Bar, on the Objects tab, click Misc, River.
2. Under **Spline Parameters**, click **Edit**, and do either of the following in your level:
   - To split a river apart, select the desired point and click **Split**.
   - To merge two rivers together, select the end point of one river and the start point of another river. Then click **Merge**.

**Adding Waterfalls**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A waterfall is a natural feature to add to cliffs or when a river changes elevation or course. Waterfalls are placed as 2D decals in the terrain.

**To add a waterfall**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the Material Editor, select a suitable texture asset.
3. Under **Material Settings**, select the **Waterfall** shader.
4. Under **Texture Maps**, place the texture in the alpha channel of the **Diffuse** texture map.
5. Expand **Diffuse\Oscillator** and adjust parameter values to produce a realistic animation effect for the texture.
6. Under **Shader Params**, adjust **Foam** parameters as needed.
7. Adjust other material settings and shader parameters as needed.
8. Place the waterfall in your level, clicking to create a simple geometry that follows the terrain.
9. Apply water (rain) particle effects if desired.

**Adding Water Puddles**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To create realistic water puddles and water rifts, use non-tiling textures that can be placed as decals. While water puddles could be created as a water volume, using decals is less demanding on resources. For more information on decals, see Working with Decals (p. 1752).

For proper blending between the water puddle and the terrain, use an alpha channel with a smooth gradient so it fades into the terrain and the transition won't be noticeable.

**To add a water puddle**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the Material Editor, select a suitable material asset.
3. Under \Lighting Settings\Specular, enter 85, 85, 85
4. In your level, click to place the puddle.
5. In the Rollup Bar, on the Objects tab, click Custom, GameVolume.
7. In your level, click boundary points around the puddle. Double-click the last point to complete the enclosure.

Adding Fog Beneath Water

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add realistic-looking fog beneath water surfaces. For more information about Lumberyard's fog system, see Fog Systems (p. 1822).

To add fog beneath water
1. In your level, click to select the water volume entity in which you want to add fog.
2. In the Rollup Bar, on the Objects tab, click Area, WaterVolume, WaterVolume Params, and modify the following parameters as needed.

   **FogDensity**
   Specifies how dense the fog appears.

   **FogColor**
   Sets the fog color.

   **FogColorMultiplier**
   Defines how bright the fog color is.

   **FogColorAffectedBySun**
   Enables the Setting Sun Parameters and Console Variables (p. 1266) Sun color parameter value to affect fog color.

   **FogShadowing**
   Enables the surface of water to receive shadows. You can control the shadow darkness. Valid values are 0–1.
   - For this parameter to function, the console variable r_FogShadowsWater must be set to 1.
   - **FogShadowing** is only available when the Config Spec setting in Lumberyard Editor is set to Very High.

   In addition, if the VolFogShadows property is enabled in the Terrain\Environment panel in Rollup Bar, shadow darkness is automatically set to full. However, the fog above the water will have volumetric shadowing.
CapFogAtVolumeDepth

If false, continues to render fog below the specified river depth.

Advanced Water Volume Parameters

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following advanced parameters apply to water volumes.

**To set advanced Water Volume parameters**

1. In the Rollup Bar, on the **Objects** tab, click **Area**.
2. Under **Object Type** click **WaterVolume**.
3. Under **WaterVolume Params\Advanced**, adjust the following parameter values as needed:

   **FixedVolume**
   
   Traces a ray down to find a 'vessel' entity and 'spill' the requested amount of water into it. For static entities, it attempts to boolean-merge any surrounding static that intersects with the first vessel (use the **No Dynamic Water** flag on brushes that do not need that).

   **VolumeAccuracy**
   
   Water level is calculated until the resulting volume is within this (relative) difference from the target volume (if set to 0 it runs up to a hardcoded iteration limit).

   **ExtrudeBorder**
   
   Extrudes the border by this distance. This is particularly useful if wave simulation is enabled as waves can raise the surface and reveal open edges if they lie exactly on the vessel geometry.

   **ConvexBorder**
   
   Takes convex hull of the border. This is useful if the border would otherwise have multiple contours, which areas do not support.

   **ObjectSizeLimit**
   
   Only objects with a volume larger than this number takes part in water displacement (set in fractions of FixedVolume).

   **WaveSimCell**
   
   Size of cell for wave simulation (0 means no waves). Can be enabled regardless of whether FixedVolume is used.

   **WaveSpeed**
   
   Sets how “fast” the water appears.

   **WaveDamping**
   
   Standard damping.
Creating Terrain

WaveTimestep
This setting may need to be decreased to maintain stability if more aggressive values for speed are used.

MinWaveVel
Sleep threshold for the simulation.

DepthCells
Sets the depth of the moving layer of water (in WaveSimCell units). Larger values make waves more dramatic.

HeightLimit
Sets a hard limit on wave height (in WaveSimCell units).

Resistance
Sets how strongly moving objects transfer velocity to the water.

SimAreaGrowth
If changing water level is expected to make the area expand, the wave simulation grid should take it into account from the beginning. This sets the projected growth in fractions of the original size. If wave simulation is not used, this setting has no effect.

Copying and Moving Terrain Areas

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can copy and move areas or sections of terrain, including vegetation, water, and other objects in your level. You can also rotate sections of terrain.

To copy or move a section of terrain

1. In Rollup Bar, on the Terrain tab, click Move Area.
2. Click Select Source and then click in the level to define the volume that is copied or moved.
3. Click Select Target and click in the level to define the target volume location.
4. Adjust the values of the following parameters as needed.

Sync Height
Sets the Z position of the source and target volumes to the same value.

Target Rotation
Rotates the source volume counterclockwise by the selected amount when moved to the target location.

DymX, Y, Z
Changes the dimension of the source volume.
Only Vegetation

Moves or copies only vegetation and other objects and not the terrain itself.

Only Terrain

Moves or copies just the terrain and not vegetation or other objects.

5. Click Copy or Move.

Blending Environment Probes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use an Environment Probe (p. 587) component to achieve the right visual quality for a space. Environment probes help to determine proper reflections, ambient diffuse values, particle diffuse values, and shadow colors.

Each environment probe in a scene represents ambient lighting information. The probe data does not change even when other lights in the scene move or change in brightness. This can result in an unnatural appearance. For example, the sun sets but objects are still brightly lit by an environment probe used for daytime lighting.

To achieve convincing transitions in lighting:

- Create multiple environment probes that represent different lighting conditions.
- Write a script that blends and puts them in sync with the scene's dynamic lights.

For example, as the sun moves from noon to dusk, a script takes brighter environment probes and blends them gradually to darker environment probes. A realistic full day-to-night cycle can require eight or more probes.

Environment probes have a Probe Fade property that fades out the environment probe. You can change the Probe Fade property with Script Canvas, Lua, and the Track View. For this property, you can specify a value between 0.0 and 1.0, which represents a percentage of ambient lighting. For example, in a scenario with two overlapping environment probes, the higher priority probe (Probe A) normally hides the lower priority probe (Probe B). Only Probe A lights objects in that area. However, if you set Probe A’s Probe Fade value to 0.5, then 50% of the ambient lighting comes from Probe A and the rest comes from Probe B.

To add an environment probe to your scene, see Working with Components (p. 479) and Environment Probe (p. 587). You can also find detailed, step-by-step instructions at Lighting the Environment in the Amazon Lumberyard Getting Started Guide.

Topics

- Dimmer and Night Light Example (p. 1252)
- Day to Night Cycle Example (p. 1256)

Dimmer and Night Light Example

The following example scenario demonstrates how to dim a lamp. The room has a dimmer lamp and a night light. As the lamp dimms, the ambient lighting provided by the environment probes must dim as well. The room does not completely darken, because of the night light.
To set up the lights, environment probes, and script

1. In Lumberyard Editor, do the following:
   a. Create an entity (p. 463) named **lamp**.
   b. Add the **Point Light (p. 706)** component to the entity.
   c. To assign a high brightness, specify the following property values:
      - **Cast shadow spec**: Low
      - **Diffuse multiplier**: 4
      - **Max Distance**: 10

2. Create another entity named **nightlight** and do the following:
   a. Add the **Point Light (p. 706)** component to the entity.
   b. To assign a low brightness, specify the following property values:
      - **Cast shadow spec**: Low
• **Diffuse multiplier:** 1
• **Max Distance:** 10

3. Create two environment probes (p. 587). Put them at the same location and make them the same size. Name one `probe_light` and the other `probe_dark`.
   a. For `probe_light`, specify the **Sort Priority** (p. 587) property to 1. This is the environment probe that fades in and out.
   b. For `probe_dark`, keep the default value for **Sort Priority** (p. 587). The value should be 0.

4. In the following step, you need to bake the `probe_light` entity. Baked means to store information about the lighting in the environment probe.

   To bake the `probe_light`, do the following:
   a. Select the `probe_light` entity.
   b. In the **Entity Inspector**, under **Cubemap generation** (p. 589), click **Generate**.
      After the cubemap is generated, the **Add Bounce** button replaces the **Generate** button.
   c. Click **Add Bounce** to rebake the probe with bounced light.
   d. **Hide** (p. 465) the `probe_light` entity. This prevents the `probe_light` entity's lighting from getting baked into the `probe_dark` entity.

5. To bake the `probe_dark` entity, do the following:
   a. Select and **hide** (p. 465) the `lamp` entity. This prevents the `lamp` entity's light from getting baked into the `probe_dark` entity.
   b. Select the `probe_dark` entity.
   c. In the **Entity Inspector**, under **Cubemap generation** (p. 589), click **Generate**.
      After the cubemap is generated, the **Add Bounce** button replaces the **Generate** button.
   d. Click **Add Bounce** to rebake the probe with bounced light.

6. **Show** (unhide) (p. 465) the `probe_light` and `lamp` entities.

7. Create an entity and do the following:
   a. Add the **Lua Script** (p. 683) component to the entity.
   b. In the **Lua Script** component, for the **Script** property, click the (...) icon, and then navigate and select the `RoomLights.lua` file. See `RoomLights.lua` (p. 1255).

   **Note**
   To create the script file, copy and paste the code into a text file. Rename the file extension to `.lua` and save it in your project directory.

8. In the **Lua Script** component, for the **LightEntity** property, click the target icon and in the viewport, select the `lamp` entity. You can also use the **Entity Outliner** to select the `lamp` entity.

   The `lamp` entity appears in the **LightEntity** box.

9. In the **Lua Script** component, for the **ProbeEntity** property, click the target icon and in the viewport, select the `probe_light` entity.

   You can also use the **Entity Outliner** to select the `probe_light` entity.

The **Lua Script** component should look like the following example:
Example

See the following `RoomLights.lua` script.

```lua
local RoomLights = {
    -- Defines properties that are exposed in the Entity Inspector window
    Properties = {
        Speed = 1.0,              -- Use to modify the speed of the light cycle
        LightEntity = EntityId(), -- Set this to a Light component that will have its intensity changed
        ProbeEntity = EntityId(), -- Set this to an Environment Probe component that will be faded out in sync with the Light component
    },
}

function RoomLights:OnActivate()
    -- Subscribes to per-frame tick updates
    self.tickBusHandler = TickBus.Connect(self);

    -- Tracks the total number of seconds that the script has been running
    self.time = 0;

    -- The level at which the light starts is its max value. Light level is reduced periodically to dim the light, and then raised back to its max level.
    self.maxLightLevel = Light.Event.GetDiffuseMultiplier(self.Properties.LightEntity);
end

function RoomLights:OnDeactivate()
    self.tickBusHandler:Disconnect();

    -- Restores entities to their default settings
    Light.Event.SetDiffuseMultiplier(self.Properties.LightEntity, self.maxLightLevel);
    Light.Event.SetProbeAttenuation(self.Properties.ProbeEntity, 1);
end

function RoomLights:OnTick(deltaTime, timePoint)
    self.time = self.time + deltaTime;

    -- Increases and decreases brightness between 0 and 1 as time passes
    local brightness = Math.LerpInverse(-1, 1, Math.Sin(self.time * self.Properties.Speed));

    -- Sets the brightness of the light
    Light.Event.SetDiffuseMultiplier(self.Properties.LightEntity, brightness * self.maxLightLevel);

    -- Sets the fade value of the probe
    Light.Event.SetProbeFade(self.Properties.ProbeEntity, brightness);
end
```
Day to Night Cycle Example

The following scenario is a more complex but commonly used example to develop a full day-to-night cycle. This example uses a concept similar to the dimmer in the Dimmer and Night Light Example (p. 1252), but instead of changing the light bulb intensity, you change the sun's position. This requires a large number of probes blending together. Also, dawn and dusk require more probes than noon and midnight. See the following procedure and script outline to get started.

To set up the day-to-night cycle probes and script

1. Create a set of entities (p. 463) to represent times. Put them in the same location and make them the same size.
2. Name the entities so that they correspond to a time on the 24-hour clock, such as probe1200 to represent noon.
   
   For example, start with 0000, 0550, 0600, 0610, 1200, 1750, 1800, and 1810. You don't need another probe at 2400 because that's the same as 0000. Notice that there are more probes clustered around dawn and dusk than noon and midnight.

   Note
   You can name these probes whatever you like, but they must end with the 4-digit time designation. The script that you apply later in this procedure looks for entities that end with four digits corresponding to the time.

3. Add (p. 479) an Environment Probe (p. 587) component to each entity.
4. Hide (p. 465) the probes so that their outputs are not captured in each other's baked lighting.
5. To bake each probe, do the following:
   
   a. Select an environment probe. You do not need to show (unhide) it for the baking process.
   b. Open the Time Of Day (p. 1270) editor.
   c. Set the Current Time to the time that corresponds to the probe name. Close the Time of Day editor.
   d. With the probe selected, click Generate to generate the cubemap.
   e. (Optional) To simulate additional light bounces, show (unhide) the environment probe and click Add Bounce.
   f. If it's not still hidden, hide (p. 465) the probe and then repeat these steps for the next probe.
6. After you set the time of day and generate (bake) the cubemap for every probe, create another entity named probe_set.
7. Select and move the probes into the probe_set entity.
8. In the **Entity Outliner**, do the following:
   
a. Select the **probe_set** entity.

b. Add the **Lua Script (p. 683)** component to the entity.

c. In the **Lua Script** component, for the **Script** property, click (...) and then navigate and select the `ProbeBlending.lua` file. See *ProbeBlending.lua (p. 1260).*

   **Note**
   
   To create the script file, copy and paste the code into a text file. Rename the file extension to `.lua` and save it in your project directory.

   
9. In the **Lua Script** component, for **Probes**, click + until the number of **EntityId** slots equals the number of probes that you have.
10. Assign each of your probes to one of the script's `EntityId` probe slots. In the **Entity Inspector**, click the target icon 🔄 next to an empty slot. In the **Entity Outliner**, select a probe. Repeat until all slots are filled.

In the following example, all of the empty slots are filled. When they are not yet selected, the `EntityId` boxes are blank.
Example

See the following ProbeBlending.lua script. To use this script, you must be in the StarterGame project.

```lua
-- This script connects a set of probes to the time-of-day cycle and blends between them as time progresses.
-- It supports an arbitrary number of probes. This means that you can use as many or as few as you need.
-- A minimum of two environment probes are required. You likely need more to get convincing results, particularly around dawn and dusk when lighting conditions change dramatically.

-- This script is provided as an example to help you get started. It is not an official feature, and is therefore not guaranteed to address every need or be completely free from defects.
```
local ProbeBlending =
{
    Properties =
    {
        Probes = { default = { EntityId(), EntityId() } }, -- Each probe entity's name must
        end in a 24-hour time code such as "envProbe1830" for a probe at 6:30pm
        ShowDebugOutput = false, -- If true, on every frame the script dumps the blend
        values for all probes to the console output
        Blend = true, -- If false, pops between probes rather than blending between them
        (mostly for demonstration purposes)
        UseToD = true, -- If true, progression is based on Time of Day (ToD). If false, an
        internal timer. Turning this off can be useful for testing purposes.
        CycleTime = 10  -- If UseToD = false, this is the number of seconds in one full
        cycle
    },
    MAX_TIME = 24.0,
    ProbeData = {} -- Will be filled with entries {Probe=, Time=} sorted by time. Or nil if
    something went wrong.
}
-- Function for sorting probes by time
function ProbeBlending.ProbeLessThan(a, b)
    return EntityId.IsValid(a.Probe) and EntityId.IsValid(b.Probe) and a.Time < b.Time or
    EntityId.IsValid(a.Probe) and not EntityId.IsValid(b.Probe);
end
-- Extracts a floating point 24-hour time value from a probe entity name. The last four
-- characters of the probe
-- name should be a 24 hour clock time value. For example, "1830" means "6:30pm" and
-- returns a value of 18.5.
-- Returns -1 if there is a problem
function ProbeBlending.ExtractTimeValue(probeName)
    if probeName:len() < 4 then
        return -1;
    elseif probeName:len() == 4 and nil == probeName:sub(-4,-1):find("[^d%d%d%d]") then -- if
    only 4 characters, they must all be digits
        return -1
    elseif probeName:len() > 4 and nil == probeName:sub(-5,-1):find("[^d%d%d%d]") then
        -- the number of time code digits shall be 4, no more, no less
        return -1
    else
        local hour = probeName:sub(-4,-3);
        local minutes = probeName:sub(-2,-1);
        return tonumber(hour) + tonumber(minutes)/60.0;
    end
end
-- This function is called upon activation to prepare self.ProbeData for processing
function ProbeBlending:ValidateAndSortProbeData()
    self.ProbeData = nil;
    if(#self.Properties.Probes < 2) then
        Debug.Error(false, "Script requires at least 2 Probes");
        return;
    end
    if(self.Properties.CycleTime <= 0 and not self.Properties.UseToD) then
        Debug.Error(false, "CycleTime must be > 0");
        return;
    end
    -- Copies the probe data into a different table where it can be easily sorted.
    local probeDataTable = {};
    for i=0,#self.Properties.Probes,1 do
local probe = self.Properties.Probes[i];
if(EntityId.IsValid(probe)) then
    local currentProbeName =
        GameEntityContextRequestBus.Broadcast.GetEntityName(probe);
    local probeTime = ProbeBlending.ExtractTimeValue(currentProbeName);
    if(probeTime < 0) then
        Debug.Error(false, "Probe Entity name " .. currentProbeName .. " does not end with a four-digit timecode");
        return;
    end
    probeDataTable[i] = {Probe=probe, Time=probeTime};
end

-- Sorts the probes according to their time codes
table.sort(probeDataTable, ProbeBlending.ProbeLessThan);

-- Further validates the data
for i=1,#probeDataTable,1 do
    local currentProbe = probeDataTable[i].Probe;
    local currentProbeTime = probeDataTable[i].Time;
    if(currentProbeTime < 0 or currentProbeTime > self.MAX_TIME) then
        Debug.Error(false, "Probe time is out of range [0," .. self.MAX_TIME .. "]");
        return;
    end
    if(i > 1) then
        local prevProbe = probeDataTable[i-1].Probe;
        local prevProbeTime = probeDataTable[i-1].Time;
        if(prevProbeTime >= currentProbeTime) then
            Debug.Error(false, "Time values must increase");
            return;
        end
    end
end

-- Saves data after it has been validated
self.ProbeData = probeDataTable;
end

function ProbeBlending:OnActivate()
    -- Subscribes to per-frame tick updates
    self.tickBusHandler = TickBus.Connect(self);
    self:ValidateAndSortProbeData();
    self.time = 0;
end

function ProbeBlending:OnDeactivate()
    self.tickBusHandler:Disconnect();
end

-- Per-frame updates are processed here
function ProbeBlending:OnTick(deltaTime, timePoint)
    local numProbes = #self.ProbeData;
    if(numProbes < 2) then return end

    local currentTime = 0;
-- Updates currentTime
if(self.Properties.UseToD) then
    currentTime = StarterGameTimeOfDayUtility.GetTimeOfDay();
else
    local rate = 1.0 / self.Properties.CycleTime;
    self.time = self.time + deltaTime * rate;
    currentTime = (self.time % 1.0) * self.MAX_TIME;
end

-- Finds pair of probes that surround currentTime
local probeIndexA = -1;
local probeIndexB = -1;
for i=1,numProbes,1 do
    local currentProbe = self.ProbeData[i].Probe;
    local currentProbeTime = self.ProbeData[i].Time;
    if(currentTime < currentProbeTime) then
        probeIndexB = i;
        if i == 1 then
            probeIndexA = numProbes;
        else
            probeIndexA = probeIndexB-1;
        end
        break;
    end
end
if(self.ProbeData[numProbes].Time <= currentTime) then
    probeIndexA = numProbes;
    probeIndexB = 1;
end

-- This first sets all attenuation values to 0 before blending in the relevant two.
for i=1,numProbes,1 do
    Light.Event.SetProbeFade(self.ProbeData[i].Probe, 0);
end

-- Calculates the blend between the two bordering probes, such that the final color
-- should be something like
-- probeA * (1-blend) + probeB * blend
local blend = 1.0;
if(probeIndexA < probeIndexB) then
    blend = Math.LerpInverse(self.ProbeData[probeIndexA].Time,
        self.ProbeData[probeIndexB].Time, currentTime);
elseif(probeIndexB < probeIndexA) then
    local passedTime = 0;
    timeBetweenProbes = self.ProbeData[probeIndexB].Time + (self.MAX_TIME -
        self.ProbeData[probeIndexA].Time);
    if(0 <= currentTime and currentTime <= self.ProbeData[probeIndexB].Time) then
        passedTime = currentTime + (self.MAX_TIME - self.ProbeData[probeIndexA].Time);
    else
        passedTime = currentTime - self.ProbeData[probeIndexA].Time;
    end
    blend = passedTime / timeBetweenProbes;
end

-- Applies the blend by setting probe fades for the two relevant probes
if(self.Properties.Blend) then
    local priorityA = Light.Event.GetProbeSortPriority(self.ProbeData[probeIndexA].Probe);
    local priorityB = Light.Event.GetProbeSortPriority(self.ProbeData[probeIndexB].Probe);
    if(priorityA == priorityB) then
        Light.Event.SetProbeFade(self.ProbeData[probeIndexA].Probe, blend);
    end
end
-- Time-adjacent probes must have different priorities in order to know which one to fade out. We'll force one to be higher
Light.Event.SetProbeSortPriority(self.ProbeData[probeIndexB].Probe, priorityB +1);
end

if (priorityA > priorityB) then
    Light.Event.SetProbeFade(self.ProbeData[probeIndexA].Probe, 1-blend);
    Light.Event.SetProbeFade(self.ProbeData[probeIndexB].Probe, 1);
else
    Light.Event.SetProbeFade(self.ProbeData[probeIndexA].Probe, 1);
    Light.Event.SetProbeFade(self.ProbeData[probeIndexB].Probe, blend);
end
else
    Light.Event.SetProbeFade(self.ProbeData[probeIndexA].Probe, 1);
    Light.Event.SetProbeFade(self.ProbeData[probeIndexB].Probe, 0);
end

-- Generates debug output
if self.Properties.ShowDebugOutput then
    local debugInfo = "Fades> ";
    for i=1,numProbes,1 do
        local currentProbe = self.ProbeData[i].Probe;
        local blendFactor = Light.Event.GetProbeFade(currentProbe);
        local currentProbeName = GameEntityContextRequestBus.Broadcast.GetEntityName(currentProbe);
        debugInfo = debugInfo .. string.format("%s: %.2f | ", currentProbeName, blendFactor);
    end
    Debug.Log(debugInfo);
end

return ProbeBlending

Working with Database Libraries

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With Lumberyard's Database View editor, you can organize and work within the libraries of your game items. You can add new or load existing libraries to your level, or remove libraries from your level. Within a library, you can organize items by customized groups, clone and edit items, assign items to selected objects, and retrieve properties from the current selection.

The Database View editor displays the following categories of libraries on its tabs:

- Entity Library
- Prefabs Library
- Vegetation (p. 1298)
• **Particles (p. 1507)**
• **GameTokens**

Using the **Database View** editor's toolbar, you can load, save, or add new libraries, or remove a library. The drop-down menu features a list of libraries that you've already opened; select an open library from this list. Use the **Add new item** tool (1) to add items to your library, **Clone library item** (2) to create a duplicate of an existing item, and **Remove item** (3) to delete the item from the library. Use **Assign item to selected objects** (4) to assign the selected item to objects currently selected in your viewport. Use **Get properties from selection** (5) to use the properties of the object selected in the viewport and apply those properties to the item selected in the item browser.

The item browser pane (A) shows the groups and items that are available in the current library. The properties pane (B) displays the properties of the item selected in the item browser. Some tabs also feature a preview pane (C).

**Adding Sky Effects**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Topics**
• **Creating a Dynamic Daytime Sky (p. 1266)**
• **Creating a Dynamic Night Sky (p. 1268)**
• **Creating Time of Day Sky Effects (p. 1269)**
Creating a Dynamic Daytime Sky

To add a dynamic daytime sky, you adjust various sun parameters, atmospheric properties, sun ray effect, and sun shadows. Dynamic skies use the SkyHDR Shader (p. 1708).

All properties and parameters in the following topics are ignored when you use a static sky (SkyBox), which uses the Sky Shader (p. 1708).

Topics
- Setting Daytime Atmospheric Effects (p. 1266)
- Setting Sun Parameters and Console Variables (p. 1266)
- Adding Sun Rays (p. 1267)
- Setting Sun Shadow Settings (p. 1267)
- Adding Cascaded Sun Shadows (p. 1268)

Setting Daytime Atmospheric Effects

To create dynamic daytime sky atmospheric effects, you modify sun and light-scattering setting that affect the appearance of distant objects, which shift in color due to atmospheric interference. These settings do not directly affect the rendering of objects or environment lighting colors and intensities.

To set daytime atmospheric effects

1. In Lumberyard Editor, choose Tools, Other, Time Of Day.
2. In the Parameters pane, for Sky Light, adjust the Sky Light Parameters (p. 1284) as needed.

Setting Sun Parameters and Console Variables

You can define how the sun appears in the daytime sky.

To set sun parameters

1. In Lumberyard Editor, choose Tools, Other, Time Of Day.
2. In the Parameters pane, for Sun, adjust the Sun Parameters (p. 1277) as needed.
You can define how the sun renders in the daytime sky.

**To set console variables for sun rendering behavior**

1. In Lumberyard Editor, in the Console pane, click the icon in the bottom left.
2. In the Console Variables window, search for and set the following console variables to control the sun's rendering behavior. For more information, see Using the Console Window (p. 210).

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_Sun</td>
<td>Activates or deactivates the sun's light source.</td>
</tr>
<tr>
<td></td>
<td>0 = Deactivate</td>
</tr>
<tr>
<td></td>
<td>1 = Activate</td>
</tr>
<tr>
<td>e_SunAngleSnapDot</td>
<td>Updates the cached sun direction if the dot product (cached sun direction and real-time sun direction) is less than the specified value for this console variable.</td>
</tr>
<tr>
<td>e_SunAngleSnapSec</td>
<td>Determines how often, in seconds, to update the cached sun direction.</td>
</tr>
<tr>
<td>e_SkyUpdateRate</td>
<td>Specifies the percentage of a full dynamic sky update, calculated per frame. This console variable affects how smoothly the sun moves in the sky, if the speed value in the Time of Day Editor is greater than 0. Higher values may impact rendering performance. Valid values: 0 to 100</td>
</tr>
</tbody>
</table>

**Adding Sun Rays**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can create a sun rays effect, which simulates the shafts of light that the sun produces under certain atmospheric conditions.

**To add sun rays**

1. In Lumberyard Editor, choose Tools, Other, Time Of Day.
2. In the Parameters pane, for Sun Rays Effect, adjust the Sun Ray Effects Parameters (p. 1287) as needed.
3. Close the Time Of Day editor.

**Setting Sun Shadow Settings**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.
You can define how sun shadows appear in your level.

**To set sun shadow settings**

1. In Lumberyard Editor, choose Tools, Terrain Tool.
2. Choose Environment.
3. Under EnvState, adjust the following parameters as needed:
   - **Sun shadows min spec** – Specifies the minimum system specification for casting sun shadows.
   - **Sun shadows additional cascade min spec** – Specifies the minimum system specification for rendering an additional sun shadow cascade at a larger viewing distance.

**Adding Cascaded Sun Shadows**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create multiple cascaded shadow maps for your level, which controls how sun shadows look at varying distances. The higher the cascade, the further it is away from the camera (cascade 0 is closest to the camera) and the lower the resolution of the shadows.

**To create cascaded sun shadows**

1. In Lumberyard Editor, choose Tools, Other, Time Of Day.
2. In the Parameters pane, for Shadows, adjust the Shadows Parameters (p. 1288) as needed for each shadow cascade.

**Creating a Dynamic Night Sky**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To add a dynamic nighttime sky, you adjust various horizon, moon, and stars settings. Dynamic skies use the SkyHDR Shader (p. 1708).

All properties and settings in the following topics are ignored when using a static sky (SkyBox).

**Topics**

- Setting Nighttime Atmospheric Effects (p. 1268)
- Setting Moon Parameters (p. 1269)

**Setting Nighttime Atmospheric Effects**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To add dynamic nighttime atmospheric effects, you set various horizon, moon, and star field parameters.
To set nighttime atmospheric parameters

1. In Lumberyard Editor, choose Tools, Other, Time Of Day.
2. In the Parameters pane, for Night Sky and Night Sky Multiplier, adjust the Night Sky Parameters (p. 1285) and Night Sky Multiplier Parameters (p. 1286) as needed.

Setting Moon Parameters

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can define how the moon appears in the nighttime sky.

To set moon parameters

1. In Lumberyard Editor, choose Tools, Terrain Tool.
2. Choose Environment.
3. Under Moon, adjust the following parameters as needed:
   - Latitude – Sets the latitude of the moon.
   - Longitude – Sets the longitude of the moon.
   - Size – Adjusts the size of the moon.
   - Texture – Specifies the asset for creating the moon texture.

Creating Time of Day Sky Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use time of day effects to create dynamic skies to simulate the changing lighting effects that are caused by the sun moving across the sky. You can also configure and store a complete day–night cycle of changing environment parameters to add realism to your level.

You can use the Time of Day editor and Sun Trajectory Tool to achieve these effects.

Note
All properties and parameters in the following topics are ignored when using a static sky (SkyBox).

Topics
- Setting Dawn and Dusk Effects (p. 1269)
- Setting a Day-Night Cycle (p. 1270)

Setting Dawn and Dusk Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can simulate the changing lighting effects that are caused by the sun moving across a dynamic sky. You can set sunrise time, duration of dawn, sunset time, duration of dusk, current time, and the path of the sun.

**To set dawn and dusk effects**

1. In Lumberyard Editor, choose **Tools, Other, Sun Trajectory Tool**.
2. In the **Sun Trajectory Tool**, set the following properties and parameter values as needed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time of Day</td>
<td>Sets the current time.</td>
</tr>
<tr>
<td>Sun Direction</td>
<td>Specifies the direction where the sun rises.</td>
</tr>
<tr>
<td>Dawn Time</td>
<td>Specifies the time the sun will rise.</td>
</tr>
<tr>
<td>Dawn Duration</td>
<td>Sets the transition duration of moon-to-sun lighting.</td>
</tr>
<tr>
<td>Dusk Time</td>
<td>Specifies the time the sun will set.</td>
</tr>
<tr>
<td>Dusk Duration</td>
<td>Sets the transition duration of sun-to-moon lighting.</td>
</tr>
<tr>
<td>Force sky update</td>
<td>If selected, updates the sky light calculations for each frame. If cleared, distributes calculations over several frames.</td>
</tr>
<tr>
<td>Import</td>
<td>Imports settings from a saved lighting file (.lgt).</td>
</tr>
<tr>
<td>Export</td>
<td>Exports current settings to a lighting file (.lgt).</td>
</tr>
<tr>
<td>Terrain Occlusion</td>
<td>Creates the effect of indirect lighting.</td>
</tr>
<tr>
<td>Super Sampling</td>
<td>Interpolates the pixels of indirect sampling data to eliminate hard transitions.</td>
</tr>
</tbody>
</table>

3. Close the **Sun Trajectory Tool** editor.

**Setting a Day-Night Cycle**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Time of Day Editor to configure changes to environment parameters over time to mimic a day-night lighting cycle. The Time of Day Editor uses a 24-hour timeline graph and a recording function to store changing environment parameter values in an XML file.

Environment parameter values that you change in the **Parameters** panel of the **Time of Day Editor** are set for the currently selected time.

The Time of Day graph shows changes to the selected parameter over time. When a parameter value is changed, the graph curve is updated for the currently selected time. You can also directly change the curve by dragging it up or down between the keyframe points. Keyframe points are displayed as yellow...
dots. You can insert new keyframe points by double-clicking the curve. To remove existing keyframe points, double-click the keyframes (yellow dots) themselves. Lumberyard interpolates parameter values for times that lie between keyframe points.

**Example**

The following is an example graph in the Time of Day Editor.

To configure a day-night cycle

1. In Lumberyard Editor, choose **Tools, Other, Time Of Day**.
2. In the **Parameters** pane, adjust the parameter value for each cycle that you want to create, and then do the following:
   a. Click the red button to start recording.
   b. In the **Time Of Day Tasks** pane, under **Time**, set the **Current Time** to apply the parameter value. The graph reflects the new value at the specified time.
   c. Set a new parameter value and current time value pair. Repeat as many times as needed to get a realistic change over time for the parameter.
   d. Click the red button to stop recording.
3. In the **Time Of Day Tasks** pane, complete the following tasks as needed to export, import, and play a time-of-day (day–night) cycle.
### Parameter | Description
---|---
**Import From File** | Imports cycle settings from an `.xml` file.
**Export To File** | Exports cycle settings to an `.xml` file.
**Reset Values** | Resets all parameters to their default values.
**Expand All** | Expand all parameters.
**Collapse All** | Collapse all parameters.
**Current Time** | Sets the current time in the **Time of Day** editor.
**Start Time** | Sets the time to use when the game starts. This is not the same value as the current time.
**End Time** | Sets the time to use when the game ends. If you set the end time to 23.59, the time loops and starts the next cycle when the day is over.
**Play Speed** | Sets the speed at which time advances in the cycle.
**Play** | Starts or resumes the playback of the cycle in the **Time of Day** editor. If the current time value is not within the start and end times, playback begins at the specified start time.
**Stop** | Stops the playback of the cycle in the **Time of Day** editor.
**Force Sky Update** | Updates the sky lighting calculations in each frame. If deselected, calculations are distributed over several frames. The effect may not be visible for some time.

4. Close the **Time of Day** editor.

### Creating Static Skyboxes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Static skies use the **Sky Shader** (p. 1708) and a skybox, which is a cube that has textures on five sides (excluding the bottom). This allows you to simulate the sky in your level. Static skies cannot use dynamic or animated **Time of Day** effects, HDR settings, and sun or moon parameters.

If you want to create a static sky, create a skybox material and then apply the material to the skybox.

**Topics**
- Creating Skybox Materials (p. 1273)
- Applying Skybox Materials to a Skybox (p. 1274)
- Changing Skybox Parameters (p. 1276)
- Switching Skyboxes with the Script Canvas Editor (p. 1276)
Creating Skybox Materials

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Skyboxes in Lumberyard are 5-sided. Sides 1 through 4 are the sides of the box, and side 5 is the top of the box. Skyboxes in Lumberyard do not have a bottom.

The sides of the box are represented by three textures that follow a specific naming convention:

1. skybox_12.tif
2. skybox_34.tif
3. skybox_5.tif

The suffix in the name indicates which sides of the box that the texture is mapped to.

You can use any image editing software to create a skybox texture. We used Adobe Photoshop to create this example and followed these specifications:

- The source sky image is authored at 8192 x 1024.
- The image uses 16 bits per channel to help preserve smooth gradients in the sky.
- The image is divided into four sections that are 2048 x 1024 each (for illustration purposes).

The numbers represent each side of the skybox that the texture is mapped to. Sides 1 and 2 will become skybox_12.tif and sides 3 and 4 will become skybox_34.tif.

The top of the example sky is a separate texture. The numbers represent which edge of the texture to map to which side. This is important for you to manage texture seams. You must name this 2048 x 2048 texture skybox_5.tif.

When you combine sides 1 and 2 into a single texture (skybox_12.tif), you must flip side 2 horizontally. When you combine sides 3 and 4 into a single texture (skybox_34.tif), you must flip side 4 horizontally. This results in two textures that are 2048 x 2048 each.
To prepare the skybox textures for Lumberyard

1. Navigate to the directory with your newly created textures.
2. Right-click each texture and choose RC Open Image.
   
   **Note**
   
   If you do not see RC Open Image, you must install RC Shell Commands using the Lumberyard Setup Assistant. For more information, see Using Lumberyard Setup Assistant to Set Up Your Development Environment (p. 16).

3. In the texture dialog box, under Preset, choose SkyboxHDR from the drop-down list. Click OK. This will create a text file (imagename.exportsettings) that Lumberyard uses to compile the texture correctly.

4. If you use source control, you must check in all images and .exportsettings files. If you do not check in these files, your scene will not render similarly for each person on your project.

Applying Skybox Materials to a Skybox

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Do the following to apply a skybox material to a skybox.

**To apply a skybox material to a skybox**

1. In Lumberyard Editor, choose Tools, Material Editor.
2. In the Material Editor, select an existing skybox material from the tree pane or create a new material.
4. Under **Texture Maps**, for the **Diffuse** texture, choose one of the three skybox textures that you created.

5. In Lumberyard Editor, choose **Tools, Rollup Bar**.

6. In the **Rollup Bar**, on the **Terrain** tab, click **Environment**.

7. Under **SkyBox**, click the browse (..) button to choose a new material for the **Material** parameter.

8. In the **Material Editor**, choose a material that uses the **Sky** shader.

9. Return to the **SkyBox** settings in the **Rollup Bar**, and click the assign (<) button to apply the selected material to the skybox.

10. (Optional) Configure the following parameters:

    - **Material low spec** – Select the skybox material to display at low resolution.
    - **Angle** – Specify the angle to rotate the skybox.
    - **Stretching** – Specify the amount to stretch the skybox texture to reduce the horizon line.

11. Close the **Material Editor**.
Changing Skybox Parameters

Do the following to modify the parameters of the dynamic sky in your level.

To change dynamic sky parameters
1. In Lumberyard Editor, choose Tools, Other, Time of Day.
2. In the Parameters pane, specify changes to the Advanced Parameters (p. 1288).

Switching Skyboxes with the Script Canvas Editor

You can use the Script Canvas editor to create a script that switches the skybox material.

To switch skyboxes with a script
1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the Script Canvas editor, choose File, New Script.
3. Right-click the canvas, search and then select the following nodes:
   - On Graph Start
   - Load by Name (p. 2586)
   - Set Skybox Material (p. 2575)
4. For your script, do the following:
   a. Select the Out pin for On Graph Start and drag to connect it to the In pin for Load By Name.
   b. For the Material Name parameter, specify the path to the material to use for the skybox. For more information, see Finding the Material Name (p. 2679).
   c. Select the Out pin for Load By Name and drag to connect it to the In pin for Set Skybox Material.
   d. Select the Material pin for Load By Name and drag to connect it to the Material pin for Set Skybox Material.
Example

5. Save the script and attach it to an entity with the Script Canvas (p. 818) component.
6. Close the Script Canvas editor.

Time of Day Parameters

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Set the following parameters to customize your sun, fog, and shadow settings.

**Topics**
- Sun Parameters (p. 1277)
- Fog Parameters (p. 1278)
- Volumetric Fog Parameters (p. 1281)
- Sky Light Parameters (p. 1284)
- Night Sky Parameters (p. 1285)
- Night Sky Multiplier Parameters (p. 1286)
- Cloud Shading Parameters (p. 1287)
- Sun Ray Effects Parameters (p. 1287)
- Advanced Parameters (p. 1288)
- Filters Parameters (p. 1288)
- Depth of Field Parameters (p. 1288)
- Shadows Parameters (p. 1288)

**Sun Parameters**

<table>
<thead>
<tr>
<th>Sun</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun color</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Sets the color (RGB values) of the sun. You can use this parameter along with Sun intensity to provide the preferred scene luminance.</td>
</tr>
</tbody>
</table>
## Sun Parameters

<table>
<thead>
<tr>
<th>Sun</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun intensity (lux)</td>
<td>float</td>
<td>0.00 to 550000.00</td>
<td>Sets the brightness of the sun. This value is multiplied by the sun color to yield the overall color.</td>
</tr>
<tr>
<td>Sun specular multiplier</td>
<td>float</td>
<td>0.0 to 4.0</td>
<td>Controls the brightness and intensity of the sun on specular materials in your scene.</td>
</tr>
</tbody>
</table>

## Fog Parameters

<table>
<thead>
<tr>
<th>Fog</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color (bottom)</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the bottom color for the fog.</td>
</tr>
<tr>
<td>Color (bottom) multiplier</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Specifies the intensity of the bottom fog color. This value is multiplied by the top fog color to set the brightness of the bottom fog color. You can set this value to 3 for midday and add three keys to the timeline with the values set at 0.02. This creates a localized orange bloom around the sun as it descends.</td>
</tr>
<tr>
<td>Height (bottom)</td>
<td>float</td>
<td>-5000 to 30000</td>
<td>Specifies a reference height for the vertical fog gradient. This is the height at which the fog color reaches the specified color at the top. For fog density, this value is the height at which the vertical density falloff reaches the specified density.</td>
</tr>
<tr>
<td>Density (bottom)</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the fog density at the bottom. Values greater than 0 or less than 1 cause the fog to gradually fall off.</td>
</tr>
<tr>
<td>Color (top)</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the color of the fog component that produces halos around the sun and sunlight scatter.</td>
</tr>
<tr>
<td>Color (top) multiplier</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Specifies the intensity of the top fog color. This value is multiplied by the bottom fog color to set the brightness of the top fog color.</td>
</tr>
</tbody>
</table>
| Height (top)       | float      | -5000 to 30000    | Sets the reference height for the vertical fog gradient. For fog color, this value is the height at which the specified color reaches at the top. For fog density, this value is the height.
### Adding Sky Effects

#### Fog

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density (top)</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the fog density at the top. You can set the top density to a higher value than the bottom density. Doing so would reverse the vertical falloff and produce thick fog in the sky and clear views at the bottom. You can also set the top and bottom density values to be equal. Volumetric fog computations treat a level as a continuous unbound volume. If you specify a density greater than 0 at the specified top height, the fog won't suddenly stop at that height. Instead the fog will continue to fall off gradually. The same action is true for bottom boundary or density values that are less than 1.</td>
</tr>
<tr>
<td>Color height offset</td>
<td>float</td>
<td>-1.0 to 1.0</td>
<td>Offsets the height of the vertical fog gradient between the top and bottom colors.</td>
</tr>
<tr>
<td>Color (radial)</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the color of the fog component that produces halos around the sun and sunlight scatter.</td>
</tr>
<tr>
<td>Color (radial) multiplier</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Specifies the multiplier of the fog color component. Radial fog is more noticeable as the light intensity decreases, so you can decrease the multiplier towards the end of the timeline. Radial fog is also applied to the moon at night, so you can create two keys on the timeline and set the value to 0.</td>
</tr>
<tr>
<td>Radial size</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the size of the radial fog glow (bloom around the sun), perpendicular to the camera. This value can also enhance effects such as the colors around the sun at sunrise or sunset.</td>
</tr>
<tr>
<td>Radial lobe</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the size of the radial fog glow towards the camera or how much the radial fog is affected by distance. Higher values create a foggier scene. Lower values affect only the horizon.</td>
</tr>
</tbody>
</table>
## Adding Sky Effects

### Fog

<table>
<thead>
<tr>
<th>Fog</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final density</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Sets the maximum density that the fog can reach. This enables the sky, horizon, and other bright, distant objects to be visible through dense fog. Setting this value too low can compromise depth perception and result in implausible visuals and apparent artifacts, especially when moving the camera.</td>
</tr>
<tr>
<td>clamp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global density</td>
<td>float</td>
<td>0.0 to 100.0</td>
<td>Sets the density of the global fog. Higher values create denser fog.</td>
</tr>
<tr>
<td>Ramp start</td>
<td>float</td>
<td>0.0 to 30000.0</td>
<td>Specifies the distance from the camera at which the fog starts to render at 0 density. This allows you to remove the fog around the camera and fade in at a specified distance.</td>
</tr>
<tr>
<td>Ramp end</td>
<td>float</td>
<td>0.0 to 30000.0</td>
<td>Specifies the distance from the camera at which the fog ceases to render at 0 density. This allows you to remove the fog around the camera and fade out at a specified distance.</td>
</tr>
<tr>
<td>Ramp influence</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies how much the ramp values affect the rendering of the fog.</td>
</tr>
<tr>
<td>Shadow darkening</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Controls the appearance of the fog in shadow areas. Specifies how much the fog color is darkened per pixel based on the volumetric shadow value per pixel. This value is applied after calculating the darkened fog color using the sun and ambient darkening factor. See the next two parameters.</td>
</tr>
<tr>
<td>Shadow darkening sun</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies how much the sun influences the radial fog color.</td>
</tr>
<tr>
<td>Shadow darkening ambient</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies how much the environment influences the ambient fog color height gradient.</td>
</tr>
</tbody>
</table>
## Fog

<table>
<thead>
<tr>
<th>Fog</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow range</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Sets the distance that the volumetric shadows are rendered until 10% (0.1) of the level's clipping plane distance is reached. Lower values produce more accurate results; however, the shadows are not rendered as far as with higher values.</td>
</tr>
</tbody>
</table>

### Volumetric Fog Parameters

<table>
<thead>
<tr>
<th>Volumetric fog</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height (bottom)</td>
<td>float</td>
<td>-5000.0 to 30000.0</td>
<td>Specifies a reference height for the vertical fog gradient. This is the height at which the fog color reaches the specified color at the top. For fog density, this value is the height at which the vertical density falloff reaches the specified density.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifies the fog density at the bottom. Values greater than 0 or less than 1 cause the fog to gradually fall off.</td>
</tr>
<tr>
<td>Height (top)</td>
<td>float</td>
<td>-5000.0 to 30000.0</td>
<td>Sets the reference height for the vertical fog gradient. For fog color, this value is the height at which the specified color reaches the top. For fog density, this value is the height at which the vertical density falloff reaches the specified density.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifies the fog density at the top. You can set the top density to a higher value than the bottom density. Doing so would reverse the vertical falloff and produce thick fog in the sky and clear views at the bottom. You can also set the top and bottom density values to be equal. Volumetric fog computations treat a level as a continuous unbound volume.</td>
</tr>
<tr>
<td>Density (top)</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the fog density at the top. You can set the top density to a higher value than the bottom density. Doing so would reverse the vertical falloff and produce thick fog in the sky and clear views at the bottom. You can also set the top and bottom density values to be equal. Volumetric fog computations treat a level as a continuous unbound volume. If you specify a density greater than 0 at the specified top height, the fog won't suddenly stop at that height. Instead the fog will continue to fall off gradually. The same action is true for bottom boundary or density values that are less than 1.</td>
</tr>
</tbody>
</table>
## Adding Sky Effects

### Volumetric fog

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global density</td>
<td>float</td>
<td>0.0 to 100.0</td>
<td>Sets the density of the global volumetric fog. Higher values create denser fog.</td>
</tr>
<tr>
<td>Ramp start</td>
<td>float</td>
<td>0.0 to 30000.0</td>
<td>Specifies the distance from the camera at which the fog starts to render at 0 density.</td>
</tr>
<tr>
<td>Ramp end</td>
<td>float</td>
<td>0.0 to 30000.0</td>
<td>Specifies the distance from the camera at which the fog ceases to render at 0 density.</td>
</tr>
<tr>
<td>Color (atmosphere)</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the fog color for the atmosphere. For example, you can set the fog color to light blue (140, 230, 255) for midday, darker blue (90, 148, 164) for sunset or sunrise, and white for night time.</td>
</tr>
<tr>
<td>Anisotropy (atmosphere)</td>
<td>float</td>
<td>-1.0 to 1.0</td>
<td>Adjusts the anisotropy for sun atmosphere scattering. Valid values: -1 to 1 Values less than 0 will shift the fog in the opposite direction of the sun. Values greater than 0 create a consistent appearance of atmospheric fog.</td>
</tr>
<tr>
<td>Color (sun radial)</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the color of the glow around the sun.</td>
</tr>
<tr>
<td>Anisotropy (sun radial)</td>
<td>float</td>
<td>-1.0 to 1.0</td>
<td>Adjusts the anisotropy for sun radial scattering. Values less than 0 will shift the glow effect to the opposite side of the sun. Values greater than 0 make the radial color visible even when looking away from the sun.</td>
</tr>
<tr>
<td>Radial blend factor</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Blends the sun radial color with the atmosphere color. Valid values: 0 to 1. Set the value to 0 to turn off radial fog.</td>
</tr>
<tr>
<td>Volumetric fog</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Radial blend mode</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Adjusts the blend mode for blending between atmosphere fog and radial fog. Set the value to 0 to use additive blending. Set the value to 1 to use linear interpolation.</td>
</tr>
<tr>
<td>Color (entities)</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the global fog color for volumetric lights, excluding the sun.</td>
</tr>
<tr>
<td>Anisotropy (entities)</td>
<td>float</td>
<td>-1.0 to 1.0</td>
<td>Adjusts the appearance of volumetric fog entities based on the viewing angle in relation to the sun. 0 = isotropic</td>
</tr>
<tr>
<td>Range</td>
<td>float</td>
<td>0.0 to 8192.0</td>
<td>Adjusts the maximum radius of volumetric fog. The default setting is 64.</td>
</tr>
<tr>
<td>In-scattering</td>
<td>float</td>
<td>0.0 to 100.0</td>
<td>Specifies how much light (including sun) is scattered by fog. Higher values create a foggier scene (fog density remains unchanged) with bigger and brighter glow effects.</td>
</tr>
<tr>
<td>Extinction</td>
<td>float</td>
<td>0.0 to 100.0</td>
<td>Specifies how much light the fog absorbs. Higher values create a thick, atmosphere effect that is tough for sun light to penetrate.</td>
</tr>
<tr>
<td>Analytical fog visibility</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Adjusts the global visibility of analytical fog. Set the value to 0 to hide analytical volumetric fog. Set the value to 1 to display analytical volumetric fog.</td>
</tr>
<tr>
<td>Final density clamp</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Sets the maximum density that the fog can reach. This enables the sky, horizon, and other bright, distant objects to be visible through dense fog. Setting this value too low can compromise depth perception and result in implausible visuals and apparent artifacts, especially when moving the camera.</td>
</tr>
</tbody>
</table>
## Sky Light Parameters

<table>
<thead>
<tr>
<th>Sky Light</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun intensity</td>
<td>R, G, B</td>
<td>0 to 254</td>
<td>Sets the illuminance of the sun and uses an RGB sun color value to compute the atmosphere color. You can use this parameter in conjunction with Sun color to provide desired scene luminance. For bright sunlight, you can use an average illuminance value of 120000 lux. The intensity of the sun decreases as it nears the horizon, so you can add keys to the timeline to lower the intensity value appropriately. To accurately render shadows at night time, you can add keys to the timeline and set the value to 5.</td>
</tr>
<tr>
<td>Sun intensity multiplier</td>
<td>float</td>
<td>0.0 to 1000.0</td>
<td>Sets the brightness of the sun. The brightness is multiplied by the sun intensity to yield the overall color. You can use this parameter in conjunction with Sun color multiplier to provide desired scene luminance. Higher values result in brighter skies. Lower values simulate an eclipse.</td>
</tr>
<tr>
<td>Mie scattering</td>
<td>float</td>
<td>0.0 to 1000.0</td>
<td>Controls mie scattering, which is caused by pollen, dust, smoke, water droplets, and other particles in the lower portion of the atmosphere. Mie scattering occurs when the particles that cause the scattering are larger than the wavelengths of radiation that are in contact with them. Mie scattering is responsible for the white appearance of clouds. Higher values create a hazy sky. Lower values create a clear sky. For a balanced sky, you can set this value to 4.8.</td>
</tr>
<tr>
<td>Rayleigh scattering</td>
<td>float</td>
<td>0.0 to 1000.0</td>
<td>Controls rayleigh scattering, which is sunlight scattering from atmospheric gases. Rayleigh scattering occurs when the particles that cause the scattering are smaller than the wavelengths of radiation that are in contact with them. As the wavelength decreases, the amount of scattering increases. Rayleigh scattering is responsible for the blue appearance of the sky.</td>
</tr>
</tbody>
</table>
## Adding Sky Effects

### Sky Light

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sky Light</strong></td>
<td></td>
<td>Higher values create a red-yellow sky. Lower values create a blue sky. For a blue daytime sky and red-yellow sunset sky, you can use the default value of 2.0.</td>
</tr>
<tr>
<td><strong>Sun anisotropy factor</strong></td>
<td>float</td>
<td>Controls the sun's apparent size. As this value approaches -1.0, the sun's disk becomes sharper and smaller. Higher values create a larger, fuzzier disk. For a balanced size, you can set this value to -0.995.</td>
</tr>
<tr>
<td><strong>Wavelength (R)</strong></td>
<td>float</td>
<td>Sets the hue (RGB values) of the atmosphere. You can create different atmospheres by adjusting the color values and gradients. This can be particularly useful with rayleigh scattering, when you choose a sun intensity of pure, bright white.</td>
</tr>
<tr>
<td><strong>Wavelength (G)</strong></td>
<td>float</td>
<td>Sets the hue (RGB values) of the atmosphere. You can create different atmospheres by adjusting the color values and gradients. This can be particularly useful with rayleigh scattering, when you choose a sun intensity of pure, bright white.</td>
</tr>
<tr>
<td><strong>Wavelength (B)</strong></td>
<td>float</td>
<td>Sets the hue (RGB values) of the atmosphere. You can create different atmospheres by adjusting the color values and gradients. This can be particularly useful with rayleigh scattering, when you choose a sun intensity of pure, bright white.</td>
</tr>
</tbody>
</table>

### Night Sky Parameters

<table>
<thead>
<tr>
<th>Night Sky</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizon color</strong></td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the horizon color of the night sky gradient. The RGB value is scaled by the multiplier. You can make a light pollution effect more pronounced by setting this value to a dark blue color (20, 36, 51).</td>
</tr>
<tr>
<td><strong>Zenith color</strong></td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the zenith color of the night sky gradient. The RGB value is scaled by the multiplier. You can make the night sky gradually appear darker further</td>
</tr>
</tbody>
</table>
### Adding Sky Effects

<table>
<thead>
<tr>
<th>Night Sky</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Night Sky</td>
<td></td>
<td></td>
<td>from the horizon by setting this value to black.</td>
</tr>
<tr>
<td>Zenith shift</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Sets the transition for the two colors in a night sky gradient. Smaller values shift the transition towards the bottom. Larger values shift the transition towards the top. You can create a smooth transition by setting this value to 0.8.</td>
</tr>
<tr>
<td>Star intensity</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Sets the overall brightness of the stars. Star flickering is by design and cannot be controlled. You can make the stars bright for night time by setting this value to 0.01 between 00:00 – 06:00 and 18:00 – 23:59.</td>
</tr>
<tr>
<td>Moon color</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the moon's emissive color. The RGB value is scaled by the multiplier. You can make the moon a less saturated blue by setting this value to 51, 58, 65. You can create a light blue transition color by creating two keys at 07:00 and 17:00 and setting the value to 200, 228, 255. Setting this value to 0 removes the moon texture.</td>
</tr>
<tr>
<td>Moon inner corona color</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the color of the moon's inner corona (glow around the moon). The RGB value is scaled by the multiplier.</td>
</tr>
<tr>
<td>Moon inner corona scale</td>
<td>float</td>
<td>0.0 to 2.0</td>
<td>Specifies the size and blurriness of the moon's inner corona. Smaller values create a bigger, blurry corona. Larger values create a smaller, focused corona.</td>
</tr>
<tr>
<td>Moon outer corona color</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Specifies the color of the moon's outer corona. The RGB value is scaled by the multiplier.</td>
</tr>
<tr>
<td>Moon outer corona scale</td>
<td>float</td>
<td>0.0 to 2.0</td>
<td>Specifies the size and blurriness of the moon's outer corona. Smaller values create a bigger, blurry corona. Larger values create a smaller, focused corona.</td>
</tr>
</tbody>
</table>

### Night Sky Multiplier Parameters

<table>
<thead>
<tr>
<th>Night Sky Multiplier</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizon color</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the multiplier for the Night Sky's Horizon color.</td>
</tr>
</tbody>
</table>
## Adding Sky Effects

### Night Sky Multiplier

<table>
<thead>
<tr>
<th>Night Sky Multiplier</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zenith color</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the multiplier for the Night Sky's Zenith color.</td>
</tr>
<tr>
<td>Moon color</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the multiplier for the Night Sky's Moon.</td>
</tr>
<tr>
<td>Moon inner corona color</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the multiplier for the Night Sky's Moon inner corona color.</td>
</tr>
<tr>
<td>Moon outer corona color</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Specifies the multiplier for the Night Sky's Moon outer corona color.</td>
</tr>
</tbody>
</table>

### Cloud Shading Parameters

<table>
<thead>
<tr>
<th>Cloud Shading</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun contribution</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Specifies how much the sun affects the cloud brightness.</td>
</tr>
<tr>
<td>Sun custom color</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Sets the RGB sun color.</td>
</tr>
<tr>
<td>Sun custom color multiplier</td>
<td>float</td>
<td>0.0 to 16.0</td>
<td>Sets the brightness of the sun, which is multiplied by the sun custom color.</td>
</tr>
<tr>
<td>Sun custom color influence</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Sets the degree to which the color of the sun contributes to the color of the clouds.</td>
</tr>
</tbody>
</table>

### Sun Ray Effects Parameters

<table>
<thead>
<tr>
<th>Sun Ray Effects</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun shafts visibility</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Sets the visibility of the sun shafts. Higher values accentuate the shadow streaks that are caused by the sun light penetrating objects.</td>
</tr>
<tr>
<td>Sun rays visibility</td>
<td>float</td>
<td>0.0 to 10.0</td>
<td>Sets the visibility of the sun rays. Higher values create brighter rays around the sun.</td>
</tr>
<tr>
<td>Sun rays attenuation</td>
<td>float</td>
<td>0.0 to 10.0</td>
<td>Sets the length of the sun rays. Higher values create shorter rays around the sun.</td>
</tr>
<tr>
<td>Sun rays suncolor influence</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Sets the degree to which the color of the sun contributes to the color of the sun rays.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set this value to 1 to use the color of the sun for the sun rays.</td>
</tr>
</tbody>
</table>
Adding Sky Effects

### Sun Ray Effects

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun rays custom color</td>
<td>R,G,B</td>
<td>Specifies a custom color for the sun rays. To use this parameter, you must</td>
</tr>
<tr>
<td></td>
<td>0 to 254</td>
<td>set the Sun rays sun color influence parameter to a value greater than 0.</td>
</tr>
</tbody>
</table>

### Advanced Parameters

<table>
<thead>
<tr>
<th>Advanced</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skybox multiplier</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Controls the brightness of a static skybox. Does not affect a dynamic sky.</td>
</tr>
</tbody>
</table>

### Filters Parameters

<table>
<thead>
<tr>
<th>Filters</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grain</td>
<td>float</td>
<td>0.0 to 8.0</td>
<td>Controls the strength of the grain filter to apply to the final image.</td>
</tr>
<tr>
<td>Photofilter color</td>
<td>R,G,B</td>
<td>0 to 254</td>
<td>Sets the RGB of the color filter to apply to the final image.</td>
</tr>
<tr>
<td>Photofilter density</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Controls the strength of the color filter to apply to the final image.</td>
</tr>
</tbody>
</table>

### Depth of Field Parameters

<table>
<thead>
<tr>
<th>Depth of Field</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus range</td>
<td>float</td>
<td>0.0 to 10000</td>
<td>Specifies the distance at which the background starts to lose focus and become blurry.</td>
</tr>
<tr>
<td>Blur amount</td>
<td>float</td>
<td>0.0 to 1.0</td>
<td>Controls the strength of blur in areas that are out of focus.</td>
</tr>
</tbody>
</table>

### Shadows Parameters

<table>
<thead>
<tr>
<th>Shadows</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade N: Bias</td>
<td>float</td>
<td>0.0 to 10.0</td>
<td>Sets the amount to move the shadow cascade away from the shadow-casting object.</td>
</tr>
</tbody>
</table>
You can create realistic-looking skies by setting sun, moon, atmospheric, and time-of-day effects. You can create two types of skies: dynamic and static.

Dynamic skies use the SkyHDR Shader (p. 1708) to render realistic effects.

The primary tools used to create different sky effects for your level are the Time of Day Editor and the Sun Trajectory Tool, as shown below:

<table>
<thead>
<tr>
<th>Shadows</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade N: Slope</td>
<td>float</td>
<td>0.0 to 500.0</td>
<td>Adjusts the gradient (slope-based) bias used to compute the shadow bias. Higher values reduce shadows that are cast from an object with a high light angle. For a more realistic effect, set the value between 32 and 64. Slope bias has little to no impact on performance.</td>
</tr>
<tr>
<td>Shadow jittering</td>
<td>float</td>
<td>0.0 to 10.0</td>
<td>Specifies shadow sharpness. Higher values may impact performance.</td>
</tr>
</tbody>
</table>
Adding Weather Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard offers a variety of realistic weather effects for your level environment.

Topics
- Adding Wind Effects (p. 1290)
- Adding Clouds (p. 1293)

Adding Wind Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create realistic wind effects in your level environment.

Topics
- Adding Global Wind (p. 1291)
- Adding Ocean Wind (p. 1291)
- Creating Wind Areas (p. 1292)
- Adding Localized Wind (p. 1292)
Adding Global Wind

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Global wind and breezes affect everything in your level, such as all vegetation. Here’s how to set them up:

To set global wind parameters
1. In the Rollup Bar, under Terrain, click Environment.
2. Under the EnvState section, adjust values of the following parameters:
   - Wind vector – Speed and wind direction vector. Positive x values are east; positive y values are north.
   - Breeze generation – Enables breezes.
   - Breeze strength – Controls the intensity of the breeze.
   - Breeze movement speed – Controls the velocity of the breeze. Use it to produce short, rapid gusts of wind.
   - Breeze variation – Varies breeze speed, strength, and size.
   - Breeze life time – Sets the duration of each breeze, in seconds.
   - Breeze count – Sets the number of breezes generated per instance.
   - Breeze spawn radius – Radius of breeze travel.
   - Breeze spread – Determines the degree of variation in breeze direction.
   - Breeze radius – Sets the radius of breeze influence.

Adding Ocean Wind

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can simulate realistic wind and wave effects for the ocean in your level.

To set ocean wind parameters
1. In Lumberyard Editor, choose Tools, Rollup Bar.
2. In the Rollup Bar, on the Terrain tab, under Terrain, click Environment.
3. Under **OceanAnimation**, adjust the following parameters:
   - **Wind Direction** – Specifies the direction to push the water texture crawl in radians. Valid values: 0.0 to 6.28.
   - **Wind Speed** – Specifies how frequently the texture water crawl updates.
   - **Wave Frequency** – Sets the frequency of waves. Smaller values mean fewer, longer waves (deep ocean depth). Larger values mean more, shorter waves (shallow ocean depth).
   - **Wave Height** – Sets wave height in meters by means of vertex displacement.

### Creating Wind Areas

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the AWS Game Tech blog to learn more.

Wind areas define a location within which objects experience wind. If no direction is set, wind moves omnidirectionally from the center of the wind area.

**To create a wind area**

1. In the **Rollup Bar**, under **Objects**, click **Entity**.
2. Under **Browser**, expand **Physics** and double-click **WindArea**.
3. Drag to place the entity in your level. A bounding box with direction areas appears.
4. Under **Entity Properties**, adjust values of the following parameters:
   - **Active** – Enables or disables wind inside the area.
   - **AirDensity** – If greater than 0, causes objects moving through the air to slow down.
   - **AirResistance** – If greater than 0, causes lightweight objects to experience buoyancy.
   - **Ellipsoidal** – Specifies an ellipsoidal drop off in air speed.
   - **FalloffInner** – Sets the distance at which distance-based air speed begins to drop off.
   - **Speed** – Sets the wind speed.
   - **Dir** – Sets the wind direction.
   - **Size** – Sets the size of the wind area.

### Adding Localized Wind

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the AWS Game Tech blog to learn more.
Localized wind is used to simulate wind from a specific object, such as a fan or jet exhaust. You set up localized wind with the wind entity.

**To set localized wind parameters**

1. In the **Rollup Bar**, under **Objects**, click **Entity**.
2. Under **Browser**, expand **Physics** and double-click **Wind**.
3. Drag to place the entity in your level at the desired location.
4. Under **Entity Properties**, adjust the following parameters:
   - **FadeTime** – Enables or disables fade time.
   - **vVelocity** – Sets the wind strength and direction.

### Adding Clouds

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
[Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can create realistic-looking clouds in your level that move, cast shadows, and that objects can fly through.

To add clouds to your level, you add the **Sky Cloud** component to an entity. With the **Sky Cloud** component, you can define the cloud's shape, its volume, how fast it moves, the dimensions of its loop box, and more.

See the following sections to set other cloud effects, such as cloud shading, cloud shadows, and 2D distance clouds.

**Topics**

- Placing Distance Clouds (p. 1293)
- Adding 3D Cloud Shadows (p. 1295)
- Setting Cloud Shading Parameters (p. 1298)

### Placing Distance Clouds

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
[Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

Distance clouds are two-dimensional clouds that are placed at a great distance, such as on the horizon or at high altitudes. Distance clouds are suitable for creating a realistic distant sky scene without causing rendering performance issues. Distance clouds do not move and cannot be flown through.
Example Distance Clouds

To place distance clouds

1. Create an entity. (p. 463)
2. Add (p. 479) the Mesh (p. 684) component to the entity.
3. In the Mesh component, for Mesh asset, click (...).

4. Select a mesh asset file and click OK.
Adding 3D Cloud Shadows

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

3D clouds don't actually cast real-time shadows. Instead a moveable texture is imposed on the entire level, creating the illusion that the clouds cast shadows.
Example 3D Cloud Shadows

To add 3D cloud shadows

1. In the Rollup Bar, under Terrain, click Environment.
2. Under CloudShadows, click Cloud shadow texture and the folder icon.
3. In Preview, select an asset.
4. Drag the shadow to the desired location in your level.

5. Under **CloudShadows**, adjust the following parameters:
   
   - **Cloud shadow speed** – Sets the speed that shadows move across the terrain.
   - **Cloud shadow tiling** – Sets the tiling multiplier of the shadow texture.
   - **Cloud shadow brightness** – Sets the brightness level of the shadow.
   - **Cloud shadow invert** – Enables inverting of the cloud shadow texture.
Setting Cloud Shading Parameters

Cloud shading, unlike cloud shadows, affects the brightness and color of clouds in your level. The environment sky and sun color affect how clouds look.

To set cloud shading parameters

1. In Lumberyard Editor, choose **Tools, Other, Time Of Day**.
2. In the **Time of Day Editor**, under **Parameters**, adjust the following **Cloud Shading** properties.
   - **Sun contribution** – Specifies how much the sun affects the cloud brightness.
   - **Sun custom color** – Sets the RGB sun color.
   - **Sun custom color multiplier** – Sets the brightness of the sun, which is multiplied by the sun custom color.
   - **Sun custom color influence** – Sets the degree to which the color of the sun contributes to the color of the clouds.

Adding Vegetation

You can add realistic trees, bushes, grasses, and other vegetation to your Lumberyard terrain.

Topics

- Vegetation Best Practices (p. 1299)
- Vegetation Recommendations (p. 1299)
- Vegetation Texture Mapping (p. 1299)
- Adding Trees and Bushes (p. 1300)
- Adding Grass (p. 1300)
- Adding Vegetation Bending Effects (p. 1302)
- Vegetation Parameters (p. 1311)
Elephant

Adding Vegetation

- Vegetation Debugging (p. 1313)
- Using SpeedTree 8 for Lumberyard (p. 1313)

Vegetation Best Practices

Keep in mind the following best practices, recommendations, and guidelines when you add vegetation to your terrain level.

- Manually place vegetation to get the most control and best results.
- To save memory, place grass manually.
- Keep the polygon count for grass blades as low as possible.
- Do not exceed a diameter of 8 meters for grass patches. This size provides a balance between performance and coverage.
- Grasses and small plants do not require specular or opacity texture maps. For more information, see Working with Textures (p. 1732).
- Set the Opacity texture at a much lower resolution than the other maps.
- Use a Glossiness value of 8 or above for realistic results.
- Use the automerged method to apply wind bending effects to grass.
- Use a maximum of 72 bones per tree for touch bending.

Vegetation Recommendations

The following settings are recommended when creating vegetation in your DCC tool.

<table>
<thead>
<tr>
<th>Vegetation</th>
<th>Polygon Range</th>
<th>Texture Size</th>
<th>Proxies</th>
<th>Material IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>0-300</td>
<td>512x512</td>
<td>Bending</td>
<td>Grass, grass proxy</td>
</tr>
<tr>
<td>Bushes</td>
<td>300-600</td>
<td>1024x1024</td>
<td>Bending, collision</td>
<td>Leaf, leaf proxy</td>
</tr>
<tr>
<td>Small Trees</td>
<td>600-1000</td>
<td>(2) 1024x1024</td>
<td>Bending, collision **</td>
<td>Trunk, leaf, leaf proxy</td>
</tr>
<tr>
<td>Large Trees</td>
<td>(2) 1024x1024</td>
<td>Bending, collision ***</td>
<td>Trunk, leaf, leaf proxy, trunk proxy</td>
<td></td>
</tr>
</tbody>
</table>

** Smaller breakable tree trunks are physicalized.

*** Larger non-breakable tree trunks are not physicalized.

Vegetation Texture Mapping

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Vegetation gets its appearance from texture mapping. Trees use two different sets of textures maps, one for leaves and branches and one for the trunk. Normal and specular maps can have a gloss map in the alpha channel.
The texture map you use depends on the type of vegetation:

**Grass** – Diffuse map only

**Leaves and branches (trees or bushes)** – Diffuse, specular, normal, and opacity maps

**Tree trunks** – Diffuse, specular and normal maps

Vegetation placement on a terrain texture layer is based on the pivot point of the vegetation object. Bigger vegetation objects might overlap with other terrain texture layers. This is most obvious if you have two different materials touching, like grass and mud.

### Adding Trees and Bushes

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add realistic trees and bushes to your terrain in your environment level. You must add trees and bushes manually.

**To add trees or bushes**

1. In the Rollup Bar, on the Objects tab, click Geom Entity.
2. Under Browser, select the desired vegetation.
3. Drag to place the tree or bush in your level.

### Adding Grass

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add realistic grass to your terrain in your environment level. You can drag to place and quickly paint the entire terrain, or manually click clump-by-clump to provide the most control and best results.

**Topics**

- Adding Grass Manually (p. 1300)
- Painting to Add Grass (p. 1301)

### Adding Grass Manually

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Although you can paint in your terrain to add grass quickly, the manual approach saves memory and results in better control and a more realistic effect.

**To manually add grass**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. Expand the left tree and select a suitable asset.
3. Under **Material Settings**, select the **Vegetation** shader.
4. Under **Shader Generation Params** select **Grass**.
5. Modify other settings and parameter values for the desired effect.
6. Click **Assign Object to Item**. Close Material Editor.
7. If necessary, depending on your terrain, in the **Rollup Bar**, on the **Terrain** tab, select the **AlignToTerrain** check box.
8. Click to place grass in your level and repeat as needed.

**Note**

When you add or move grass, it may sporadically jump around. This happens if you move vegetation to a location that is too dense to accommodate it. When this occurs, the vegetation moves to its last position and is outlined in red. You can then move it elsewhere or delete it.

**Painting to Add Grass**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

You can drag the mouse to quickly paint all terrain in your level with grass. This method of placing vegetation is controlled by the texture layer that the vegetation object is associated with. Painted vegetation is visible wherever the texture layer appears. This is a quick way to automatically cover a huge area with grass.

**Note**

Painting a level with grass consumes 8 MB of memory cache.

**To add grass by painting**

1. In the **Rollup Bar**, on the **Terrain** tab, click **Vegetation**.
2. Under **Vegetation**, click **Add Vegetation Object**.
3. In **Preview**, select a suitable asset.
4. Click to place the grass in your terrain.
5. Under the **Use on Terrain Layers** parameter, select the check box for your asset. The terrain should now be covered with the grass object.

### Adding Vegetation Bending Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard provides three methods for adding realistic bending motions to vegetation:

- **Touch (Collision) Bending** – bending effects for larger vegetation caused by players brushing against or colliding with branches
- **Detail (Wind) Bending** – physically accurate wind effects for larger vegetation defined by using vertex colors and environment wind parameters
- **Automerged (Wind) Bending** – physically accurate wind effects for grass defined by vegetation and environment wind parameters

You can use touch and detail bending effects together. For example, a player can brush against a branch that is also swaying in the breeze. Use automerged bending by itself for objects like grass.

From a performance standpoint, detail bending is the least expensive, touch bending is more expensive, and automerged bending is the most expensive.

**Topics**

- Adding Touch (Collision) Bending Effects (p. 1302)
- Adding Detail Bending Effects (p. 1307)
- Using AutoMerged Wind Bending Effects (p. 1310)

### Adding Touch (Collision) Bending Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The touch bending technique simulates a player touching, brushing against, and interacting with vegetation. Use it for bushes, branches, flexible trees, and bigger leaves with stems.

To start implementing touch bending vegetation effects in your game, you must have mesh assets for your vegetative elements in FBX format. FBX files can be exported from most 3D modeling software packages.

For a specific vegetation element, you must have:

- The FBX file representing the vegetation asset as a mesh
- A cube-shaped proxy mesh that represents the "trigger volume" for collisions
- A skeleton (tree of joints) used to define the branches as well as the skinning data

The figure below shows how these 3 components are represented in the Lumberyard user interface after you have imported the main vegetation object mesh (as FBX).
Adding Vegetation

You can see the object's geometry, cube proxy mesh (to determine when collisions occur between the player and the object), and the skeleton used for physical behaviors when a collision occurs. Bending occurs when the player volume intersects with the mesh, and the nature of the bending (within the tree of Joints) is typically based on where the collision occurred with respect to the cube proxy mesh.

- Vegetation FBX Mesh (p. 1303)
- Cube Collision Proxy Mesh (p. 1303)
- Vegetation Skeleton (p. 1304)
- Debugging and Performance Tips (p. 1305)

Vegetation FBX Mesh

An FBX mesh can be exported into Lumberyard from many common digital content creation tools, or from an existing .fbx asset file.

For more details on working with FBX meshes in Lumberyard, read Customize FBX asset export with FBX Settings (p. 409).

Cube Collision Proxy Mesh

The cube proxy mesh is a simple primitive used to determine when a collision (a “touch”) occurs between the vegetation object and some other physical entity. You import it as an FBX mesh, and if you name this...
mesh as *touchbend* (for example, proxy_touchbend) the FBX pipeline will automatically identify this mesh as the trigger volume.

This mesh is cube-shaped and big enough to cover the main mesh, and is used at runtime as a trigger volume. Whenever a physical entity enters this volume during your game's inner loop processing, the Lumberyard engine will trigger the creation of a "skeleton" structure made of dynamic rigid bodies (segments) attached to each other via joints (specifically, D6Joints in PhysX). This structure simulates a terrain-anchored skeleton that passively reacts to the movement of physical entities pushing through it.

**Vegetation Skeleton**

For users of standard digital content creation (DCC) tools like Maya, 3D Studio, or Blender, this part of the asset is nothing more than joints organized in a parent-child relationship. This skeleton is used to skin the main mesh and define a binding pose for it.

**Limitations:**

The skeleton cannot have more than 128 joints.

**Naming Convention Requirements:**

The joints must follow the naming convention "branchN_M", where N and M are positive integers starting from number 1. Here is an example showing joint structure and naming:

![Joint Structure Diagram]

**Note**

The name of the root joint can be postfixed with "_touchbend" (for example, "branch1_touchbend") as a convenience to notify the FBX pipeline that this asset is used for touch bending instead of a traditional animated character.
Debugging and Performance Tips

In the Lumberyard Editor, you can instantiate or paint static touch bendable vegetation using the Rollup Bar (LEGACY) UI.

The steps shown above are no different from painting regular vegetation objects. Just make sure that the chosen CGF file comes from an FBX asset exported as Touch Bendable Geometry.

Alternatively, you can use Dynamic Vegetation planting with help from the Vegetation Gem.

Improving performance

There is a set of CVARs that can be used to fine-tune the performance of touch bendable vegetation.

<table>
<thead>
<tr>
<th>CVAR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_CullVegActivation</td>
<td>Maximum distance, in meters, from the camera for a touch bendable vegetation node to be considered for bending. The default value is 50 meters. In many cases, this value can be reduced to 12m and still provide good results and higher performance. Setting this value to 0 indicates that there is no activation limit.</td>
</tr>
</tbody>
</table>
## Adding Vegetation

<table>
<thead>
<tr>
<th>CVAR</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_FoliageBranchesTimeout</td>
<td>Lifetime, in seconds, of touch bending physics simulation after the last time a vegetation object was touched by another physical entity. The default value is 4 seconds.</td>
</tr>
<tr>
<td>physx_Debug</td>
<td>Set this CVAR to 1 to enable the drawing of PhysX rigid bodies and trigger volumes when debugging. This allows you to view the interactions during run time and check for unexpected behaviors during object interactions.</td>
</tr>
</tbody>
</table>

The example pictures below show the trigger volume at the time it is first touched by a spherical collider. In the next frame, a tree of capsules is created dynamically and its realtime movement is fed into the renderer for skinning. This tree of capsules will remain alive in PhysX memory for e_FoliageBranchesTimeout seconds unless another collision occurs during that window, in which case the window will reset.
Adding Detail Bending Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Detail bending refers to the procedural movement of larger foliage caused by wind or other similar effects. You control the motion by the use of vertex colors in your DCC tool.

When you use detail bending, make sure the distribution of polygons on foliage geometry is regular and properly tessellated. Otherwise you may see visual artifacts.

Contents
- Defining Vegetation Vertex Colors with a DCC Tool (p. 1307)
- Setting the Detail Bending Parameter (p. 1308)

Defining Vegetation Vertex Colors with a DCC Tool

Vegetation objects use vertex colors to specify detail bending. Lumberyard uses all three RGB channels to control the movement of the geometry. Use your DCC tool to edit and view each channel separately.

When painting the vertices, begin with the blue channel so that you can block in the overall motion of the vegetation object. Next, paint the red channel with the vertex color mode set to additive. Use a low opacity so that you can slowly build up the effect. Finally, edit the green channel with vertex color mode set to additive. Use a low opacity so that you can slowly build up the effect.

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB Values</th>
<th>Bending Influence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>255/0/0</td>
<td>Irregular bending at the outsides – movement of smaller shapes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random sinusoidal noise with a high frequency.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The red channel works only if the surface normals of the vegetation face horizontally. If the leaf is flat and all normals face up, the red channel has no effect.</td>
</tr>
<tr>
<td>Green</td>
<td>0/255/0</td>
<td>Delays the start of the movement – used to create variations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random sinusoidal noise with a high frequency.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Values less than 0, 255, 0 delay the start of the sinusoidal noise.</td>
</tr>
<tr>
<td>Blue</td>
<td>0/0/255</td>
<td>Bends the leaves up and down – movement of the big shapes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random sinusoidal noise with a low frequency.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Values less than 0, 0, 255 introduce a greater amount of the noise. Only the blue channel can stop all movement. You can use the blue channel for the unmoving parts of vegetation, such as the base of a blade of grass or where a leaf attaches to its stem. Also use the blue channel on tree trunks so that they sway in the wind.</td>
</tr>
</tbody>
</table>

Once you have painted all of the channels, export your asset as an .fbx file. Your result might look like the following.
Example

![Image of vegetation with vertex colors]

**Setting the Detail Bending Parameter**

Once you define the vertex colors, you must set the *Bending* parameter for detail bending. This value controls the procedural bending for vegetation. Its value ranges from 0 to 100. A value of 0 means no bending effect. A value of 100 indicates the maximum effect when receiving environmental wind.

For more information, see, *Adding Global Wind (p. 1291).*

**To set the vegetation detail bending parameter**

1. In Lumberyard Editor, choose **Tools**, **Terrain Tool** and then choose **Vegetation**.
2. In the **Vegetation** section, click the first icon to add a vegetation object.
3. In the dialog box, select your vegetation object and click **OK**.
4. In the **Vegetation** section, select the asset that you want to modify, and then set the *Bending* parameter from 0.0 to 100.
5. Select your vegetation object, click **Paint Objects**, and then in the viewport, click to place the vegetation object.

6. To modify the vegetation material, in the **Vegetation** section, right-click the asset and choose **Go To Object Material**.

   This opens the **Material Editor**, where you can edit the material and its **Vegetation Shader** (p. 1712).

7. In the **Material Editor**, select the material that is applied to the asset on which you painted vertex colors in your DCC.

   Under **Shader Generation Params**, select the following options:
   
   - **Detail bending**
   - **Leaves** or **Grass**, depending on your use

   After you check these options, several new **Shader Params** appear.

8. Adjust the following parameters to achieve the effect that you want.
- **Bending branch amplitude** – Affects the amplitude of the vertices painted blue.
- **Bending edges amplitude** – Affects the amplitude of the vertices painted red.
- **Detail bending frequency** – Affects the frequency of the detail bending.

The following example uses the `am_aspen_01_group.cfg` file in Starter Game.

**Example**

<table>
<thead>
<tr>
<th>Bending = 2.0</th>
<th>Bending = 6.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image 1" /></td>
<td><img src="image2.png" alt="Image 2" /></td>
</tr>
</tbody>
</table>

**Using AutoMerged Wind Bending Effects**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Automerged vegetation has physically accurate wind motions that are defined by wind environment properties and various vegetation properties. It is recommended for use with grass only.

Automerged vegetation reduces the number of draw calls while still allowing you to add any amount or size of grass patches to the terrain. It merges multiple meshes within several sectors as long as they are using the same material and texture. You can paint single grass blade objects on the terrain as well as on brushes in different heights independently while they get merged into larger chunks in real time.

When AutoMerged is enabled, touch bending, vertex colors, and detail bending settings are all ignored, and vegetation movement is defined solely by the AutoMerged parameters. For more information, see Adding Touch (Collision) Bending Effects (p. 1302) and Adding Detail Bending Effects (p. 1307).

**To enable AutoMerged vegetation and set parameters**

1. In the Rollup Bar, on the Terrain tab, click Vegetation, Add Vegetation Object.
2. Expand Objects tree and select the grass object you want to modify.
3. Select the AutoMerged check box, and adjust the following parameter values:
   - **Stiffness** – Defines the stiffness of the vegetation
   - **Damping** – Specifies the amount of damping on the bending motion
   - **AirResistance** – Specifies the amount of bending similar to the Bending parameter used for Setting the Detail Bending Parameter (p. 1308).

The four AutoMerged parameters together define the amount and type of bending motions the vegetation object displays in reaction to wind and breezes. For more information, see, Adding Global Wind (p. 1291).

**Vegetation Parameters**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following vegetation parameters can be accessed in the Terrain, Vegetation panel in the Rollup Bar for a previously selected vegetation object. You can adjust parameters for one or more selected objects.

Unless otherwise noted, parameters apply to newly added and placed vegetation assets only.

**Size**

Changes the size of newly placed vegetation objects. Use this to uniformly scale the vegetation, where 1 represents 100%.

**SizeVar**

Changes the limit of size changes for a set of newly placed vegetation objects of a single kind. Keep Size at 1 and set SizeVar to 0.2 to get a nice variation in sizes.

**RandomRotation**

Randomly rotates objects while you paint new vegetation objects. To create a more natural look and distribution, you can set up a RandomRotation in the vegetation objects when you paint them. This feature works only with the Paint Object tool.
**AlignToTerrainCoefficient**

Points the vegetation object away from the ground. When this effect is applied, vegetation on cliffs points away from the surface instead of growing straight up.

**UseTerrainColor**

Makes the individual object receive the color of the underlying terrain for a better match. Use this option to blend the grass with the underlying terrain color. You can also use this setting on other objects, but it works best with grass. This effect is especially useful for making grass appear to fade in the distance.

**AllowIndoor**

Enables the vegetation to be rendered within vis areas.

**Bending**

This value controls the bending deformation of the vegetation objects. It ranges from 0 to 100, with 0 representing no bending effect and 100 the maximum effect. This effect is based on the amount of environment wind (WindVector) in the level.

**GrowOnBrushes**

Controls the placement of objects on brushes.

**GrowOnTerrain**

Controls the placement of objects on terrain. Useful if you want them placed only on brushes.

**AutoMerged**

Enables AutoMerged system on this object. For more information, see Using AutoMerged Wind Bending Effects (p. 1310).

**Stiffness**

Controls the stiffness of selected vegetation and how much it reacts to physics for AutoMerged vegetation.

**Damping**

Determines how responsive the vegetation is to physics damping for AutoMerged vegetation.

**AirResistance**

Degree that vegetation resists air movement (wind). Similar to the Bending setting but specifically designed for AutoMerged vegetation.

**Pickable**

Allows the player to pick up the object.

**Density**

Adjusts the distance between individual objects that you create while painting new vegetation. The density setting ranges from 0 to 100. If your density setting is bigger than your brush radius, the vegetation will not be created, so always make sure you have a suitable brush radius.

**ElevationMin**

Limits the minimum height at which you can paint vegetation objects. For painting underwater vegetation, set this value to lower than the ocean; 0 is a safe option.

**ElevationMax**

Limits the maximum height at which you can paint vegetation objects.
SlopeMin

Limits the minimum angle of the terrain on which you can paint vegetation objects. 255 equals 90 degrees. When you specify a SlopeMin value higher than 0, you can no longer place objects on flat grounds.

SlopeMax

Limits the maximum angle of the terrain on which you can paint vegetation objects. 255 equals 90 degrees. When you specify a SlopeMax lower than 255, you can no longer place objects on very steep areas.

CastShadow

Makes the object cast a shadow based on the minimum selected Config Spec setting. For example, High won't work on Low or Medium specs.

Vegetation Debugging

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Branches and tree trunks can be broken upon collision.

e_vegetation 1 | 0

Enables and disables rendering of the vegetation. 1 = on, 0 = off.

e_MergedMeshesDebug 1

Displays statistics on global memory consumption of vegetation objects placed in the level.

e_MergedMeshesDebug 2

Displays vegetation in the cells that form the merged meshes. They are color coded over distance. Red boxes should be displayed only around the player (the cell the player is standing in and the surrounding eight cells). Beyond this, all cells should be green.

Displayed above each cell is information about the current LOD step and memory consumption for the cell—this updates as you move closer and further away.

Using SpeedTree 8 for Lumberyard

SpeedTree 8 for Lumberyard is a procedural vegetation modeling program that takes advantage of the native Lumberyard vegetation tools. You can use SpeedTree 8 for Lumberyard to create and quickly iterate on trees, bushes, exotic alien foliage species, and other plant life.

The SpeedTree integration provides a simple workflow to add level of detail (LOD) and preview wind accurately with Lumberyard's wind physics integrated directly into the app.

The SpeedTree modeler exports Lumberyard meshes and materials into your game project. You can then use these meshes and materials the same way you would use any other mesh or vegetation object.
Download SpeedTree 8 for Lumberyard.

**Note**
You will need to create a SpeedTree account and activate your license in order to download SpeedTree 8 for Lumberyard.

View the SpeedTree 8 documentation.

---

**Dynamic vegetation**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

The dynamic vegetation system uses vegetation components to customize vegetation coverage for worlds of any size. To use the dynamic vegetation system, you must enable the **Vegetation** gem for your game project.

For more information, see [Enabling Gems](#).
Using combinations of components from vegetation and other categories, you can do the following:

- Create layers of background vegetation and specify local vegetation areas around environmental objects and points of interest such as buildings.
- Place vegetation using live WYSIWYG (what you see is what you get) vegetation authoring.
- Reconfigure any aspect of vegetation during any development phase without starting from scratch. All edits are nondestructive and populate rapidly throughout the level.
- Create complex ecological biomes all in one nested slice containing many layers of broad coverage for blending across areas or an entire world.
- Configure vegetation to grow only on certain surfaces. Use components to specify where vegetation can grow such as on a range of ground slope angles or at specified altitudes.

The dynamic vegetation system interacts in several ways with the static vegetation system (p. 1298). For example, you can do the following:

- Use the same assets in both dynamic and static vegetation systems
- Configure static vegetation to block dynamic vegetation

Dynamic vegetation differs from static vegetation in the following ways.

<table>
<thead>
<tr>
<th>Dynamic Vegetation</th>
<th>Static Vegetation</th>
</tr>
</thead>
<tbody>
<tr>
<td>World size</td>
<td>Limited only by Lumberyard</td>
</tr>
<tr>
<td></td>
<td>maximum world size</td>
</tr>
<tr>
<td>Templates</td>
<td>Slices</td>
</tr>
<tr>
<td>Saved as</td>
<td>Procedural mechanisms</td>
</tr>
<tr>
<td>Generated</td>
<td>In-game just-in-time placement</td>
</tr>
<tr>
<td></td>
<td>As level data</td>
</tr>
<tr>
<td></td>
<td>Maximum of 2K–4K</td>
</tr>
<tr>
<td></td>
<td>Not templatable</td>
</tr>
<tr>
<td></td>
<td>Static placement data</td>
</tr>
</tbody>
</table>

You can further modify dynamic vegetation with components from other categories. To use the full capabilities of the dynamic vegetation system, enable the following gems.
Vegetation gem (required)

Provides the dynamic vegetation system and the core vegetation components. These are the foundation for creating areas and registering them with the system.

Gradient Signal gem (dependency)

Provides gradient and gradient modifier components, which are vital to the procedure-driven methodology of dynamic vegetation. The gradient components generate various gradient signals, such as random noise and Perlin noise. The gradient modifier components modify and mix the gradient signals.

Using gradient signals with modifier components (such as positional jitter) or filter components (such as vegetation distribution) produces realistically random expressions of vegetation in the game world.

Surface Data gem (dependency)

Enables surfaces such as terrain or meshes to emit signals, or tags, that communicate its surface type. Using a vegetation surface mask filter, you can choose to grow vegetation on a particular surface by using an inclusion list or block it from growing by using an exclusion list. You could also recapture the tags as a gradient signal by using the Surface Mask Gradient component.

FastNoise gem (optional)

Provides an expressive FastNoise Gradient component that generates many procedural noise variations. In the Lumberyard Entity Inspector, the FastNoise Gradient component appears in the Gradient category. You use it like any other gradient component.

Topics

- Dynamic Vegetation Concepts (p. 1316)
- Dynamic Vegetation Procedures (p. 1319)

Dynamic Vegetation Concepts

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The dynamic vegetation system operates around the following ideas:

- **What** – To specify the type of vegetation that appears, you must have prebuilt vegetation assets, such as .cgf or .fbx files. If you don't have your own, you can use Lumberyard's vegetation assets, which are in the Starter Game project.
- **Where** – You can create small patches of flowers or populate a massive area with convincing varieties of plants and objects. To do this, use local area clusters in the foreground layer or coverage areas in the background layers, respectively.

  Using vegetation filters, you can set up rules that specify the surface angles, altitudes, distances from objects, and surface types where vegetation grows.

  Vegetation blockers block out areas of vegetation. For example, to create a border free of vegetation around a house, use a vegetation blocker that is slightly larger than the base of the house.

- **How** – Pairing vegetation modifiers with an appropriate gradient makes vegetation appear random in position, scale, rotation, and alignment to the slope. Gradients also customize how frequently each vegetation asset appears when you use multiple assets for one area.
• If – You can use surface tag emitters to tag certain types of surfaces, such as roads, rivers, oceans, and meshes. The dynamic vegetation system reads the tags to recognize these surfaces and acts on a set of rules that you create in the form of inclusion and exclusion lists. For example, to make water lilies to appear on all water surfaces, add the water volume surface tag to the inclusion list. To ensure that they never appear on roads, add the road surface tag to the exclusion list.

Example

The following image shows grass excluded from the river and road surfaces, and pebble clusters included only on road surfaces.

You can customize your vegetation in the following ways:

• Assign priorities to your vegetation areas. Priorities determine the order in which the vegetation system fills the areas.

• Use customizable gradients such as the **Perlin Noise Gradient** to mimic the types of vegetation groupings often found in nature.

• Save your customized vegetation areas as slices so that you can easily reuse them in other levels or share them with collaborators. Use slice overrides to make small or large changes to individual instances of the slice.

Dynamic Vegetation Components

The common workflow for creating a new vegetation area starts with creating an entity and adding a **Vegetation Layer Spawner** component to it. Then you add to that entity the two required components, which define the area's shape and the assets to display. From there, you can add optional components such as vegetation filters and modifiers.

Any other components that further modify the vegetation area, such as gradients, blockers, and blenders, must be contained on separate entities. These separate entities can be sibling or child entities of the entity that has the **Vegetation Layer Spawner** component.

The following table summarizes the functions of dynamic vegetation components and the components that they typically interact with. The most commonly used components are at the top of the table. Components that appear in a category, such as vegetation filters, are referenced as such rather than
individually named. The optional and incompatible columns aren't comprehensive of all components from every category, only those related to dynamic vegetation.

<table>
<thead>
<tr>
<th>Primary Component Attached to an Entity</th>
<th>Description</th>
<th>Required Components</th>
<th>Optional Components</th>
<th>Incompatible Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vegetation Layer Spawner</td>
<td>Primary component for creating a vegetation area. Determines layer priority and filter stage.</td>
<td>• Shape component (or Reference) – Defines vegetation area's size and shape &lt;br&gt; • Vegetation Asset List (or Combiner) – Specifies the assets to display in this vegetation area</td>
<td>• Vegetation Filter category (zero or more components) &lt;br&gt; • Vegetation Modifier category (zero or more components) &lt;br&gt; • Vegetation Layer Debugger component &lt;br&gt; • Vegetation Asset Weight Selector component</td>
<td>• Gradients category of components &lt;br&gt; • Gradient Modifier category of components &lt;br&gt; • Vegetation Layer Blocker, Vegetation Layer Blocker (Mesh), Vegetation Layer Blocker (Static) components &lt;br&gt; • Vegetation Layer Blender component</td>
</tr>
<tr>
<td>Gradients category (only one per entity)</td>
<td>Creates a gradient that other components such as the Vegetation Distribution Filter and the Vegetation Asset Weight Selector can reference. Provides a gradient that the referencing components use to distribute vegetation or other environmental ornaments. Gradient types include Perlin noise, white noise, image, and so on.</td>
<td>Some components in this category require another component. For example: &lt;br&gt; • The Image Gradient component requires a Gradient Transform Modifier component &lt;br&gt; • The Constant Gradient component requires no additional components</td>
<td>You can add certain Gradient Modifier components alongside the Gradient component.</td>
<td>Certain Gradient Modifier components are incompatible. For a better workflow, add Gradient Modifier components to a separate entity and then reference the Gradient entity ID.</td>
</tr>
<tr>
<td>Gradient Modifiers category (only one per entity)</td>
<td>Modifies a gradient. Reference the Gradient entity ID in the Gradient</td>
<td>Some components in this category require another component. For example:</td>
<td>None</td>
<td>• Some Gradients components</td>
</tr>
</tbody>
</table>
Lumberyard User Guide Lumberyard User Guide
Dynamic Vegetation Procedures

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Purpose</th>
<th>Components Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a Vegetation Layer (p. 1320)</td>
<td>The first basic step in creating dynamic vegetation. Establishes the size and shape of your vegetation area and adds vegetation assets.</td>
<td>Vegetation Layer Spawner, a Shape component, and Vegetation Asset List</td>
</tr>
<tr>
<td>Adding Scale, Rotation, and Position Modifiers (p. 1329)</td>
<td>Changes the look of your vegetation even more by adding scale, rotational, and positional</td>
<td>Vegetation Rotation Modifier, Vegetation Scale Modifier, and Vegetation Position Modifier</td>
</tr>
</tbody>
</table>

Dynamic Vegetation Procedures

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Creating realistic dynamic vegetation in Lumberyard starts with a few basic procedures that you can follow up with endless customizations.

You can perform these procedures in the order that they appear, which presents a basic workflow. Or you can pick and choose the procedures that you need. The examples in these procedures use assets in the Starter Game project.

The following table summarizes each documented procedure, its purpose, and the components used.
Dynamic Vegetation Procedures

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Purpose</th>
<th>Components Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saving the Vegetation Area as a Slice (p. 1334)</td>
<td>Save your vegetation as a slice so that you can place instances of it in multiple areas of your level.</td>
<td></td>
</tr>
<tr>
<td>Expanding Vegetation Coverage (p. 1336)</td>
<td>Expand your vegetation coverage to use in another area or to cover your entire level.</td>
<td>Vegetation Reference Shape or Shape component</td>
</tr>
<tr>
<td>Blocking Vegetation in Select Areas (p. 1338)</td>
<td>Create a vegetation blocker to prevent vegetation from appearing in a specific area of your level.</td>
<td>Vegetation Layer Blocker and a Shape component</td>
</tr>
</tbody>
</table>

Creating a Vegetation Layer

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Creating a vegetation layer is the first and most basic step in creating your dynamic vegetation. The following procedure uses a simple workflow and assets from the Starter Game project.

To create a vegetation layer

1. Create an entity (p. 463) and name it.
   In this example, the entity is named BasicCoverage.

2. Add (p. 479) the Vegetation Layer Spawner component to your entity.
The **Vegetation Layer Spawner** component is the core component that initializes the engine that spawns vegetation.

3. Click *Add Required Component* and choose *Vegetation Reference Shape*.

The **Vegetation Reference Shape** has no shape on its own. You must next create a child entity and add a *Shape* component, which you will reference in the **Vegetation Reference Shape**.

4. Right-click *BasicCoverage*, select *Create Child Entity*, and name it *TestBox*.
5. Select *TextBox*, click *Add Component*, and select the *Box Shape* component.
6. Adjust the size and position of the shape so that it's large enough for your purposes and intersects with the ground.

7. Select *BasicCoverage* and, in the **Vegetation Reference Shape** component, click the target symbol and select the *TestBox* entity.

8. Click *Add Required Component* and choose *Vegetation Asset List*.

The **Vegetation Asset List** component defines what to plant. This is where you specify vegetation assets.
9. In the **Vegetation Asset List** component, next to **Mesh Asset**, click **Browse (...)**.

10. In the **Search** bar, enter **grass** and select one of the grass assets in the results.
You should have a uniform grassy field with the grass in a grid formation.
Using Gradients to Create Random Distribution

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use gradients in different areas of Lumberyard, such as with audio and AI. Gradients are particularly helpful in dynamic vegetation, where they create a realistically random look in the distribution of your vegetation.

You can achieve the appearance of random distribution by using gradients to create both random selection and random placement.

Before you can complete these procedures, you must first create a vegetation layer (p. 1320).

Random selection means that the vegetation that is selected for each point on the grid is variable. Each asset's chance of being selected depends on the weight that is assigned to it. You create weight-based random selection using the **Vegetation Asset Weight Selector** component.

Random placement means that some points on the grid have vegetation on them and some don't. The **Vegetation Distribution Filter** limits the amount of vegetation that the **Vegetation Layer Spawner** component produces.

**Topics**
- Creating Weight-Based Random Selection (p. 1324)
- Random Placement Using the Vegetation Distribution Filter (p. 1328)

### Creating Weight-Based Random Selection

When you create distribution using random selection, the vegetation asset that is selected for a particular point on the grid varies. The assigned weight of each asset determines its chance of selection. In this procedure, you specify weights and link to the gradient using the **Vegetation Asset Weight Selector** component.

**Preparing the Vegetation Entity**

The vegetation entity, or the entity that contains the **Vegetation Layer Spawner** component, must contain a component that assigns values for selection of the listed assets. This procedure uses the **Vegetation Asset Weight Selector** component for that purpose. It's also helpful, though unnecessary, to add multiple assets if you haven't already. If you have only one asset listed, your one asset and bare ground are used when you link the gradient in Creating a Gradient Entity (p. 1326).

**To prepare the vegetation entity**

1. Ensure that you have three or more vegetation assets on your entity that has the **Vegetation Layer Spawner**.

   To add more vegetation assets, do the following:

   a. In the **Vegetation Asset List** component's properties, next to **Embedded Assets**, click +.

      **Note**
      When specifying your first asset, you don't need to click + because there is an empty asset displayed by default. Specify the mesh.
b. Under the blank asset, listed as `<asset name>`, next to Mesh Asset, click **Browse (…)**.

c. In the search bar, enter a search term, such as `flower`, and select a vegetation asset.

Repeat this step with different assets until you have three or more assets.
Although you have specified multiple assets, only the first asset that you added appears in the Viewport. The next several procedures show you how to distribute the vegetation so that all of them appear.

2. Click **Add Component, Vegetation Asset Weight Selector**.

   In the next step, **Creating a Gradient Entity (p. 1326)**, you link this component to the gradient, which supplies values between 0.0 to 1.0 at a given position. Assets are then mapped based on those values and each asset's weight and order values.

**Creating a Gradient Entity**

The gradient entity provides the noise signal to reference from the vegetation entity.

**To create a gradient entity**

1. **Create a child entity (p. 463)** under your vegetation area entity and select it.
2. Rename your new entity to a descriptive name, such as **Gradient**.

3. **Add (p. 479)** the component **Perlin Noise Gradient**.

   This component generates a type of noise called Perlin noise, which mimics the type of randomness found in nature.

   **Note**
   
   If you don't have the **Gradient** category in your list of components, you must enable the **Gradient** gem.

4. Add the required **Gradient Transform Modifier** component.

   This component controls how the procedural noise is generated in the world space.

5. Click **Add Required Component** and select **Vegetation Reference Shape**.

6. In the **Vegetation Reference Shape** component's properties, next to **Shape Entity Id**, click the target button.

7. In the **Entity Outliner**, select the **TestBox** entity (or the entity that contains your shape if you named it something else).

   The **Shape Entity Id** field populates with the entity name that you selected and uses the shape on that entity as its reference shape.
Linking the Gradient to the Vegetation Area

Before the gradient that you created can have any effect on the vegetation, you must reference the gradient from within the vegetation area. This means the component that you reference the gradient (in this example, the Vegetation Asset Weight Selector) in uses the gradient's information for its selection of values.

You can reference the gradient that you created in any number of components. In the procedure Adding Scale, Rotation, and Position Modifiers (p. 1329), you use the same gradient for the vegetation modifiers.

To link the gradient entity to the vegetation entity

1. Select the entity BasicCoverage.
2. On the Vegetation Asset Weight Selector component's properties, next to Gradient Entity Id, click the target.

3. In the Entity Outliner, select the Gradient entity.

   The Gradient Entity Id field populates with the entity name.

   Your vegetation area should now have variation in its vegetation selection.
Random Placement Using the Vegetation Distribution Filter

The **Vegetation Distribution Filter** component creates the look of random placement by limiting the amount of vegetation that the **Vegetation Layer Spawner** component produces.

Before completing the following procedure, you must have the following:

- At least one asset defined in your vegetation layer.
- An entity that contains a gradient component. For instructions, see the section called "Creating a Gradient Entity" (p. 1326).

**To create random distribution**

1. In the **Entity Outliner**, select the entity that contains the **Vegetation Layer Spawner** component.
2. In the **Entity Inspector**, click **Add Component** and select **Vegetation Distribution Filter**.
3. In the **Vegetation Distribution Filter** component's properties, next to **Gradient Entity Id**, click the target button.
4. In the **Entity Outliner**, select the **Gradient** entity.
5. (Optional) Adjust the values for **Threshold Min** and **Threshold Max** to specify how much of the gradient can appear in the vegetation.
Adding Scale, Rotation, and Position Modifiers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Before you can complete this procedure, you must first create a vegetation layer (p. 1320).

If you have added a gradient for random selection (p. 1324), the vegetable selection appears random, but still produces a grid-like pattern. This is because each type of vegetation is all the same size, faces the same direction, and is indeed planted on points on a grid.

To remedy this, do the following:

- **Add a scale modifier** – Varies the size of the vegetation. You can specify a range of scale, which sizes vegetation up or down while maintaining original proportions.
- **Add a rotation modifier** – Changes the direction that vegetation faces. In this procedure, you modify only the z-axis rotation.
- **Add a position modifier** – Modifies where on the grid points the plants appear. You can shift them by varying amounts on the x-, y-, and z-axes. Modifying the x- and y-axes moves the plants on the ground plane. Modifying the z-axis changes the height at which a plant sprouts (this procedure doesn't modify the z-axis).

Applying modifiers to your vegetation gives it a realistic, natural look.
Adding a Scale Modifier

A scale modifier varies the size of the vegetation.

**To add a scale modifier**

1. Select the **BasicCoverage** entity.
2. In the **Entity Inspector**, add the **Vegetation Scale Modifier** component.

   By itself, this component has no effect because there is not yet any information that tells it how to distribute the values.

3. In the **Vegetation Scale Modifier** component's properties, under **Gradient**, next to **Gradient Entity Id**, click the target.

4. In the **Entity Outliner**, select **Gradient**.

   The **Gradient Entity Id** field populates with the entity name.

   This component has a **Range Min** and **Range Max** both set at 1.0 by default, so there are no results yet.

5. Adjust the **Range Min** value to 0.4 and the **Range Max** value to 1.25.

   **Range Min** sets the scale size for the gradient signal's lowest values. **Range Max** sets the scale size for the gradient signal's highest values. Because the gradient signal varies in range from black to white, a scale value between the minimum and the maximum is applied to that vegetation instance.
Adding a Rotation Modifier

A rotation modifier varies the rotation of the vegetation.

**To add a rotation modifier**

1. Select the BasicCoverage entity.
2. In the **Entity Inspector**, add the **Vegetation Rotation Modifier** component.

   By itself, this component has no effect because there is not yet any information that tells it how to distribute the values.

3. In the **Vegetation Rotation Modifier** component's properties, under **Rotation Z, Gradient**, next to **Gradient Entity Id**, click the target.

4. In the **Entity Outliner**, select **Gradient**.

   The **Gradient Entity Id** field populates with the entity name.

Adding a Position Modifier

A position modifier shifts each instance of vegetation by an amount that the gradient determines, which removes the grid-like appearance of the vegetation.

**To add a position modifier**

1. Select the BasicCoverage entity.
2. In the **Entity Inspector**, add the **Vegetation Position Modifier** component.

   By itself, this component has no effect because there is not yet any information that tells it how to distribute the values.

3. In the **Vegetation Position Modifier** component's properties, under **Position X, Gradient**, next to **Gradient Entity Id**, click the target.
4. In the **Entity Outliner**, select **Gradient**.

   The **Gradient Entity Id** field populates with the entity name.

   The result is a slight x-axis variation in the ranges specified (the default is -0.3 to 0.3). For greater variation, modify the **Range Min** and **Range Max** values.

5. In the **Vegetation Position Modifier** component's properties, under **Position Y, Gradient**, next to **Gradient Entity Id**, click the target.
6. In the **Entity Outliner**, select **Gradient**.

   The **Gradient Entity Id** field populates with the entity name.

   In this procedure, you pass the same gradient signal to both the x and y modifiers, resulting in the same offset for both. This shifts all the vegetation in a common diagonal direction instead of varying the x from the y.

   You can override this by providing some additional values in the **Position Y** properties.

   **Note**
   You can also overcome this issue by using a gradient modifier on your existing gradient or by creating a separate gradient and linking to it.

7. Under **Position Y**, expand the **Advanced** header and check **Enable Transform**.
8. To produce a swizzling effect on the y-axis, use the following values or a variation on them.

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Translate</strong></td>
<td>0.5</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>-1.0</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Rotate</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>45.0</td>
</tr>
</tbody>
</table>

**Saving the Vegetation Area as a Slice**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Save your vegetation area as a slice so that you can easily place it in other locations or levels and further customize it.

Slices enable you to place multiple instances of the vegetation for your level. If you make a change to a slice, you can save the override to all other instances of the slice. For example, if you change the vegetation asset type from red to yellow flowers, you can choose to update all instances of the slice so that the vegetation area appears with yellow flowers.

For more information, see Working with Slices (p. 510).

To reference vegetation using slices

1. In the Entity Inspector, select the BasicCoverage entity.
2. Right-click the entity and choose Create slice, enter a name such as BasicCoverage.slice, and save it to a directory.
3. In the **Asset Browser**, navigate to the **BasicCoverage.slice** file.

4. Select and drag the file into the viewport. This creates an instance of the slice.

**Example**

Two slice instances in the viewport have the same vegetation assets and configuration.

5. Select the new instance of the **BasicCoverage** entity.

6. On the **Vegetation Asset List** component, click the **Mesh Asset** and select a yellow flower asset.

**Note**

A property that appears orange has an override. An override is a component change that is different than its source slice. For more information, see **Modifying a Slice and Saving Changes** (p. 519).

7. Right-click the property, choose **Save field override**, and choose **basiccoverage.slice**.

This updates all instances of the slice to use yellow flowers.
Expanding Vegetation Coverage

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you create a vegetation patch, you can expand it so that your vegetation covers the entire level.

To expand vegetation coverage

1. Create an entity and name it `WorldBox`.
2. Click Add Component and select the Box Shape component.
3. In the Box Shape component, specify values for the x-, y-, and z-axes to match your level. For example, if you created a level with a texture dimension of 512 x 512, specify similar values such as 512.0, 512.0, and 64.0.
4. Select the BasicCoverage entity.
5. On the Vegetation Reference Shape component, for Shape Entity Id, select the WorldBox entity.

Example

The vegetation appears for the entire level. As you move through the level, the vegetation dynamically appears.
Reusing Vegetation in Multiple Areas

After you create your vegetation entity, you can designate your customized vegetation to appear in another area. You do this by referencing another shape. For example, if you created a forest patch that includes the vegetation assets and arrangement for your level, you can create another entity with a different shape and reference this new shape for your vegetation. With this feature, you can reuse your vegetation in different areas of your level without creating a separate vegetation entity each time.

To reuse vegetation in a new area

1. In the **Entity Inspector**, select the **BasicCoverage** entity.
2. Right-click the entity, choose **Create child entity**, and name it **Cylinder(PlacementTest)**.
3. Click **Add Component** and select the **Cylinder Shape** component.
4. Select the **Cylinder(PlacementTest)** entity and move it away from the vegetation area.
5. Select the **BasicCoverage** entity.
6. On the **Vegetation Reference Shape** component, for **Shape Entity Id**, specify the **Cylinder(PlacementTest)** entity.
Dynamic Vegetation Procedures

Example

The vegetation appears in the cylinder instead of the box shape.

Blocking Vegetation in Select Areas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create areas in your level to block vegetation from appearing. For example, you can use this feature to create areas around buildings or homes where vegetation shouldn't appear.

To block vegetation in select areas

1. Create an entity and name it Blocker.
2. Click Add Component and select the Vegetation Layer Blocker component.
3. Click Add Required Component and select a shape, such as the Box Shape component.
4. In the Box Shape component properties, for Dimensions, enter values for the x-, y-, and z-axes, such as 6.0, 6.0, and 4.0.
5. Move the **Blocker** entity to the vegetation area. The entity blocks the vegetation from appearing in the specified shape.

**Example**

A box shape vegetation blocker placed in the same area as a **Mesh** component blocks vegetation within and around the tower mesh.
Actor and effects animation in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Working with animation in Lumberyard

Learn how to animate game characters and other actors in your Lumberyard scenes and cinematics, as well as how to add particle-based animation effects.

In this section:

<table>
<thead>
<tr>
<th>Topic area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character animation (p. 1340)</td>
<td>Learn how to create and animate game characters and other actors in your Lumberyard game scenes and levels using the Animation Editor and EMotionFX (EMFX).</td>
</tr>
<tr>
<td>Particle effects (p. 1504)</td>
<td>Learn how to add particle-based effects to your Lumberyard game scenes and levels using the Particle Editor.</td>
</tr>
<tr>
<td>Cinematics (p. 1583)</td>
<td>Learn how to create in-game cinematic sequences with the Track View Editor and component entities.</td>
</tr>
</tbody>
</table>

Create and animate characters

Most game projects require an animated character to move around in the environment. This may be a character that the player controls, or an AI-driven entity that interacts with the level.

Use the EMotion FX Animation Editor to animate characters in Amazon Lumberyard. To build a character, you associate one or more skinned models with an animation skeleton (built in a digital content creation tool like Maya). You then import the character into the Animation Editor and specify the animations that you want your character to have.

You can then blend animations, so that your character transitions from one animation to another, and specify the conditions when an animation occurs for a character. For example, you can specify that your
character starts in an idle position. After several seconds, the character starts to walk, run, and then slow down again until the character returns to an idle position.

In the **Animation Editor**, you can preview the animations and blends between the animations for your characters.

Lumberyard has the **FBX Settings** tool that converts static .fbx meshes, skeletons, skins, animations, and materials into Lumberyard assets. For more information, see [Customize FBX asset export with FBX Settings](p. 409).

**Note**
If you are using Lumberyard 1.10 or earlier, you can use the legacy animation system (Geppetto and Mannequin) to accomplish these tasks. For more information, see [Animation System](p. 1341) in the **Amazon Lumberyard Legacy Reference**.

**Topics**
- Animation Editor Concepts and Terms (p. 1341)
- Animation Editor User Interface (p. 1348)
- Animation Editor File Types (p. 1352)
- Getting Started with the Animation Editor (p. 1353)
- Referencing External Anim Graphs (p. 1360)
- Synchronizing Animation Graphs: Example (p. 1372)
- Animation Editor Components (p. 1398)
- Using Morph Targets to Deform Characters (p. 1399)
- Customizing State Machine Routing with Sparse Motion Sets (p. 1413)
- Animation Editor Nodes (p. 1415)
- Using Tags with Animation Graphs (p. 1446)
- Customizing EMotion FX Objects (p. 1449)
- Creating Custom Motion Events and Parameters Using C++ (p. 1451)
- Creating and Simulating a PhysX Ragdoll (p. 1458)
- Creating Additive Animations (p. 1470)
- Retargeting Motions (p. 1471)
- Creating Simulated Objects (p. 1473)
- Add Cloth Colliders to actors (p. 1489)
- Best Practices for Working with Skinned Meshes for Animation (p. 1492)

**Animation Editor Concepts and Terms**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](p. 1341) or visit the [AWS Game Tech blog](p. 1341) to learn more.

The following concepts and terms are used in the **Animation Editor**:

**Topics**
- About Animation Graphs (p. 1343)
- About Parameters (p. 1345)
- About Motion Sets (p. 1347)
- About State Machines (p. 1348)
Actor

A character with at least one bone is called an actor. An actor consists of a set of nodes in a hierarchy. Each node is a transformation (position, rotation, and scale) and can contain a mesh.

Instances of actors are called actor instances. For example, one soldier that is instanced 100 times can create an army of the same character. You can animate actor instances separately, so that each instance behaves differently. Each actor instance has unique transformations, but shares the same hierarchy as the actor from which it was instanced.

Actor files have the .actor extension (for example, hero.actor).

Motions

Motions are individual animation clips, such as walk loop, an idle motion, and so on. Motions contain transformation animation and/or morph target weight animation. A motion can contain animation data for the skeletal structure and the morph targets.

Motion files have the .motion extension (for example, Walk.motion).

Animation graphs

Animation networks are called animation graphs. Animation graphs contain the state machines, transitions, conditions, blend trees, and other nodes. Animation graphs are hierarchical.

Animation graph files have the .animgraph extension (for example, Main.animgraph).

Motion sets

A motion set contains a list of motions, where each motion has a unique string ID (for example, walk, idle, and so on). Nodes inside an animation graph can reference motions based on their string ID.

Motion sets can be hierarchical. A child set can override specific motions for a character, while sharing the rest of the motions with the parent set.

Motion set files have the .motionset extension (for example, MainSet.motionset).

Motion events

Markings at specific time values inside the motion files are called motion events (also called notifiers or notifications). A motion event has a type string (for example, "SOUND") and parameter string (for example, "Footstep").

Motion events can have a fixed time value or range. Events with a single time value are called tick events. Events with specified start and end times are called ranged events.

You can specify event presets, which are pre-setup types of events that you drag and drop into the event tracks. An event track is a group of events that you enable or disable. For example, you can add all sound effects to an event track specifically for sounds.

Motion event data is stored in the motion FBX's .assetinfo file.

Synchronization

You can use full clip-base sync or sync tracks to synchronize motion clips to keep a character's motions in sync while blending. For example, if your character is running, synchronization helps keep the right and left feet in sync.

Full clip-based sync warps the motions so that there is a constant change in playback time of the child motion.

Sync tracks use motion events, where the events mark specific moments (for example, where the right and left feet are on the floor). This system is also known as phase matching and dynamically controls the playback speed.
Floats

Floats are numbers with decimals (for example 1.35 or 1.0.) Booleans and integers are also floats, so they can be passed as weight float inputs to blending nodes. If floats are rounded, the Animation Editor always rounds them down. For example, 2.99 becomes 2.

Time

All time values and durations are in seconds. For example, you can set the transition time for 0.3 or 300 ms.

About Animation Graphs

Contents

• Animation Graph Nodes (p. 1343)

Animation graphs define the animation behavior for your game characters. Animation graphs contain the states that the character can have and define the transitions between these states. Each transition can have a set of conditions that define the logic behind the transitions.

Animation graphs contain nodes and the connections between them. These connections define how data is passed between the nodes or how transitions occur between nodes.

Animation graphs have two main node types:

• State machines
• Blend trees

Because animation graphs are hierarchical, the nodes can be nested. For example, you can have a state machine inside a state machine inside a blend tree, which contains another blend tree and state machine, and so on. The number of hierarchy levels is limitless, but as a best practice limit your hierarchy to 20 levels or fewer.

Each animation graph has one root node, which is a state machine. This root node is the default and cannot be deleted. A simple animation graph can contain one state inside this root state machine. For example, the single state can be a motion node, which outputs a pose that is applied to the character, such as an idle motion.

Before you can add nodes to an animation graph, you must create a motion set. After you create a motion set, you can create an animation graph and then assign the motion sets to the animation graph using the Resource Management pane in the Anim Graph window.

Animation Graph Nodes

In a state machine, you can add the following nodes from the Sources category:

• Blend Tree
• Entry
• Exit Node
• Hub
• Motion
• State Machine
• Bind Pose
In a blend tree, you can add other nodes from the following six categories

1. **Sources**
   - Float Constant
   - Parameters
   - Blend Tree
   - Entry
   - Motion Frame
   - Exit Node
   - Motion
   - State Machine
   - Bind Pose

2. **Blending**
   - Pose Subtract
   - Morph Target
   - Pose Mask
   - Blend N
   - Blend Two (Legacy)
   - Blend Space 2D
   - Blend Space 1D
   - Blend Two

3. **Controllers**
   - Transform
   - LookAt
   - TwoLink IK
   - AccumTransform

4. **Logic**
   - Pose Switch
   - Float Condition
   - Float Switch
   - Bool Logic

5. **Math**
   - Direction to Weight
   - Range Remapper
   - Vector3 Compose
   - Vector2 Compose
   - Vector4 Decompose
   - Vector4 Compose
   - Vector3 Math1
   - Vector3 Decompose
   - Smoothing
   - Float Math2
   - Vector2 Decompose
   - Vector3 Math2
   - Float Math1

6. **Misc**
About Parameters

Contents

- Adding Parameters to an Animation Graph (p. 1345)
- Adding a Parameter Node to a Blend Tree (p. 1346)

When you create your animation graph, you can use parameters to control how your animations transition between different states.

Each transition can have a set of conditions applied to it. These conditions define the logical rules for the transition and how animations blend together.

Each transition condition is controlled by a set of parameters. Your Lumberyard game setup sends parameter values to the animation graph. The actor reacts to the incoming parameters. The game sends the parameter values to the animation graph, which then responds to the changes automatically. For example, you can specify parameter values such as speed, direction, weapon type, and so on.

You can set this up in a game level by adding an Actor and an Animation component to an entity with the Entity Inspector.

For more information, see Animation Editor Components (p. 1398).

Adding Parameters to an Animation Graph

You can add parameters to an animation graph in the Parameters pane.

To add a parameter to an animation graph

1. In Lumberyard Editor, choose Tools, Animation Editor.
2. In the Parameters pane, click the green + icon.
3. In the Create Parameter dialog box, specify the parameter name, description, and the value type.

You can specify the following value types to provide input into animation graph nodes:

- Float (slider)
- Float (spin box)
- Boolean (checkbox)
- Tag (checkbox)
- Integer (slider)
- Integer (spin box)
- Vector2
- Vector3
- Vector3 gizmo
- Vector4
- String
- Color
- Rotation
- Group
You can name parameter types to identify the purpose of the control. For example, you can name parameters such as movement_speed, movement_direction, jumping and attacking. As an artist and game designer, you can specify the parameters that best control your animation graph.

Adding a Parameter Node to a Blend Tree

After you create your parameters in the Parameter pane, you can add a parameter node to your blend tree.

To add a parameter node to your blend tree

1. In the Animation Editor, right-click the animation graph grid and choose Create Node, Sources, Parameters.
2. In the Attributes pane, click select parameter and specify the parameter that you want.
You can rename parameter nodes and specify them to provide input to other nodes. In the following example, the `speed_parameter` node provides input to the blend tree.

About Motion Sets

A motion set is a collection of motions, where each motion refers to a specific motion file and is identified by a string ID, such as `idle_motion1`. When you create motion nodes, you specify the string ID for the motion, not the motion file itself. You can use different motion sets in combination with the same animation graph. For example, you can create an animation graph to define animation behavior for a controllable human character and apply the same animation graph to a frog. Because frog movements differ from a human character's, you specify different motion sets for the frog. You can share animation graphs for your characters; you don't need to create unique animation graphs for each character type.

A combination of an animation graph with a specified motion set being applied to a given actor instance is called an animation graph instance. Each animation graph instance has a unique set of parameter values. For example, an army of 100 soldiers is controlled by 100 different animation graph instances, which allows you to animate each soldier independently.

Motion sets can also be hierarchical. Child motion sets can override certain motions from their parents. When you apply a child motion set to a character, the character uses all the motions shared by the parent except for the motions that are specified for the child motion set. For example, you can have a
character that shares 90% of the same motions of the parent, but has custom motions specific to that character.

**About State Machines**

State machines contain a set of states that are linked together by transitions. A transition goes from one node to another node and has properties, such as the time it takes to make the transition. A blend between the outputs of both states is performed during the transition, when the animation moves from one state to another.

Transition conditions are conditions that are linked a given transition. For example, they can compare a parameter value against another value to see if the given parameter is bigger than the specified value. If the condition is met, this signals the trigger for the transition. For example, if the speed parameter is greater than 0, a character transitions from an idle to a run state. You can apply multiple conditions to a single transition. The transition occurs only when all conditions are met.

**Animation Editor User Interface**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Topics**

- Opening Animation Editor (p. 1348)
- Anim Graph Interface (p. 1349)
- Motion Sets Interface (p. 1350)
- Motions Interface (p. 1351)

**Opening Animation Editor**

You can open the Animation Editor from Lumberyard Editor.

**To open the Animation Editor**

- From Lumberyard Editor, choose Tools, Animation Editor.
On the **Animation Editor**, you can do the following:

A. From the menu bar, you can do the following:
   - Open actor files.
   - Open, create, and save your workspaces.
   - Select and deselect actor instances.
   - Change the layout view of the editor.
   - Display or hide menus for the **Animation Editor**.

B. View your character in the perspective window.

C. **Anim Graph** – View and edit your animation graph to define your character’s behavior.

D. **Anim Graph Palette** – Drag and drop nodes into the animation graph grid.

E. **Anim Graphs Navigation** – View the hierarchy of your nodes.

F. **Parameters** – View, add, edit, or delete parameters for the animation graph.

G. **Attributes** – View, add, edit, or delete attributes for the selected node in the animation graph.

H. **Resource Management** – Add, load, save, and delete animation graphs.

I. **Recorder** – View what triggers the recording and manage the playback options.

J. **Time View** – View recorded debug information and edit event tracks of the selected motion.

---

**Anim Graph Interface**

Use the animation graph to create nodes and specify parameter and values for each node.
On the animation graph, you can do the following:

A. From the menu bar, you can do the following:
   - Open and save your animation graphs.
   - Select and deselect nodes.
   - Change the layout view of the editor.
   - Display or hide windows for animation graph.

B. In the grid, create nodes and connect them with connectors.

**Motion Sets Interface**

Use the **Motion Set** pane to create motion set files and add motion files to it.
In theMotion Setspane, you can do the following:

A. **Motion Sets Management** – Add, open, save, or delete motion sets.

B. **Motion Sets** – Add, open, or delete the motion files in the selected motion set.

**Motions Interface**

Use the motions pane to add and manage motion files in a motion set.

In the**Motions**pane, you can do the following:

A. Add, save, open, or delete motions.

B. **Motion Properties** – Specify how you want the motion to animate and the play speed of the motion.
Lumberyard User Guide

Animation Editor File Types

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you import .fbx files from your DCC to Lumberyard Editor, Asset Processor creates files that you use in the Animation Editor. The following example shows how the file types are created and modified. For more information about processing files for character and animations, see Customize FBX asset export with FBX Settings (p. 409).

The following are file types used in the Animation Editor:

**Files Types**

When you import .fbx files into Lumberyard, Asset Processor generates the following file types used for the Animation Editor:

- **.actor** files are created when the .fbx file has at least one bone. You want to use your character's skinned mesh as your .actor file. The .actor file is the character that displays the animation.
- **.motion** files are created when the .fbx file with a bone has at least one keyframe. If your .fbx file has animation keyframes, a .motion file is created. Your .motion files contain the animations that are added to your motion set before you build your animation graph.
- **.mtl** files are created when the .fbx file has at least one material, which is the case for most DCC tools. If you make changes to the material in the Material Editor, the .mtl file is no longer a child of the .fbx file and the .mtl file is a sibling in the source directory of the .fbx file. You can make other changes to the .mtl file with a text editor or the Material Editor.

When a .motion file is created, an .actor file is also created. The .actor file that you want to use in the Animation Editor is generally the skinned mesh in the bind pose that you export from your DCC. Keep track of your skinned mesh that you want to use as your .actor file in the Animation Editor. You can go to the FBX Settings tool and delete the actor files that you don't need.
File Types Required for Animation Graphs

When you create your animation graph in the Animation Editor, the animation graph must have the following files:

- .actor
- .motion
- .motionset
- .animgraph

Saving Animation Editor Files

Saving your project in the Animation Editor creates a .workspace file. The workspace saves the actor, motion, motion set and animation graphs that you are using. When you open a workspace, the Animation Editor loads the files so that you can pick up where you last left off.

To save your workspace

- In the Animation Editor, choose File, and then choose one of the following:
  - Save Workspace
  - Save Workspace As

When .actor and .motion files are saved, the Animation Editor creates an .assetinfo file, alongside the source .fbx file. The .assetinfo files stores the configuration and settings for the .actor and .motion files.

Settings saved for .actor files include the actor name, motion extraction node, excluded bounds, collision mesh setup, and mirror setup.

Settings saved for .motion files include the motion extraction capture height option and motion events.

To save .actor and .motion files:

- In the Animation Editor, do one of the following:
  - Click Save All to saves any changes made to the .actor, .motion, .motionset, and .animgraph files. A dialog box prompts you to choose which files to save.
  - Click Save Workspace to saves your current workspace. If you don't have a workspace saved, a dialog box appears so that you can name your workspace and save it to your preferred directory.
  - Click Save Workspace As to saves your workspace with a different name or to another directory.
  - To save motion files individually, click the save icon in the Motions pane.
  - To save actor files individually, click the save icon in the Actor Manager pane.

Getting Started with the Animation Editor

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
See the following procedures to get started with the Animation Editor.

In this procedure, you do the following:

- Import your actor file and create a motion set to specify the motions that you want for your character.
- Create a basic animation graph using nodes.
- Create a blend tree to combine the motions and use a slider to control character movement from idle to walking and then running.

You can use the Samples Project to find the files used in this procedure. For more information, see Samples Project (p. 146).

The files for this procedure are located in the following directory:

`lumberyard_version\dev\SamplesProject\AnimationSamples\Simple_JackLocomotion`

**Topics**

- Step 1: Creating a Motion Set (p. 1354)
- Step 2: Creating an Animation Graph (p. 1356)
- Step 3: Blending the Animations (p. 1357)

### Step 1: Creating a Motion Set

In the following procedure, you import your character, Jack the robot, select the motions that you want, and then add those motions to a motion set.

**To create a motion set**

1. In Lumberyard Editor, choose Tools, Animation Editor.
2. In Animation Editor, choose Layouts, AnimGraph.
3. In the Animation Editor, choose File, Open Actor and navigate to the AnimationSamples\Simple_JackLocomotion directory.
4. Select the JackBind_ZUp.fbx file and then click OK.

Your character Jack appears in the Animation Editor.
5. On the **Motion Sets** tab, under **Motion Set Management**, click the + icon to add a motion set.

6. Select the **MotionSet0** node.

7. In the **Motion Set** pane, click the folder icon to add motions.

8. Navigate to the AnimationSamples/Simple_JackLocomotion directory, and select the following files:
   - Jack_Idle_ZUp.fbx
   - Jack_Strafe_Run_Forwards_ZUp.fbx
   - Jack_Strafe_Walk_Forwards_ZUp.fbx

9. Click **OK**.

10. In the **Motion Set Management** pane, click the **Save** icon.

11. Navigate to the /SamplesProject/AnimationSamples/Simple_JackLocomotion directory. For the file name, type **quickstart** and then click **Save** to save the quickstart.motionset file.
Step 2: Creating an Animation Graph

In the following procedure, create an animation graph and nodes.

To create an animation graph

1. On the **Anim Graph** tab, click the + icon to create an animation graph.
2. Click the **Save** icon.
3. Navigate to the `/SamplesProject/AnimationSamples/Simple_JackLocomotion` directory. For the file name, enter **quickstart** and then click **Save** to save the `quickstart.animgraph` file.
4. On the **Anim Graph** tab, right-click the grid, and then select **Create Node, Sources, Motion**.
5. Select the **Motion0** node and in the **Attributes** pane, click **Select motions**. In the dialog box, select `Jack_Idle_ZUp.fbx` and then choose **OK**.
6. Right-click the grid and then choose **Create Node, Sources, Blend Tree**.
7. From the **Motion0** node, click and drag a line to the **BlendTree0** node. A transition line with an arrow connects the nodes.
8. From the **BlendTree0** node, click and drag a line to the **Motion0** node.

![BlendTree0 and Motion0 nodes](image)

9. In the **Parameters** pane, click the + icon to create a parameter.
   a. Leave the **Value type** parameter to the default, **Float (slider)**.
   b. For **Name**, rename **Parameter0** to **speed**.
   c. Click **Create**.

10. In the animation graph, select the transition line that starts from the **Motion0** node and connects to the **BlendTree0** node.
   a. In the **Attributes** pane, click **Add condition**.
   b. In the **Select a Condition** dialog box, select **Parameter Condition** and then click **Add Condition**.
   c. In the **Attributes** pane, under **Parameter Condition**, click **Select parameter** and select **speed**.
   d. For **Test Function**, leave the default value of **param > testValue**. This means that if the speed is greater than zero, the idle motion transitions to the blend tree, and the character starts to move.

![Parameter Condition](image)

11. In the animation graph, select the transition line that starts from the **BlendTree0** node and connects to the **Motion0** node.
   a. In the **Attributes** pane, click **Add condition**.
   b. In the **Select a Condition** dialog box, select **Parameter Condition** and then click **Add Condition**.
   c. In the **Attributes** pane, under **Parameter Condition**, click **Select parameter**. Select **speed** and then click **OK**.
   d. For **Test Function**, select **param == testValue**. This means that if the speed is equal to zero, the motion transitions back to idle, and the character stops moving.

![Parameter Condition](image)

**Step 3: Blending the Animations**

In the following procedure, you use the blend tree node to build your blend tree, which blends the walk and run animations together.
To blend the animations

1. In the animation graph, double-click the BlendTree0 node.
2. Right-click the grid and choose Create Node, Sources, Motion.
3. Select the Motion1 node.
   a. In the Attributes pane, choose Select motions.
   b. In the Motion Selection Window, select jack_strafe_walk_forwards_zup and then click OK.

   The attributes for the Motion1 node should look like this:

   ![Motion1 Attributes](image)

4. In the animation graph, right-click the grid and choose Create Node, Sources, Motion.
5. Select the Motion2 node.
   a. In the Attributes pane, click Select motions.
   b. In the dialog box, select jack_strafe_run_forwards_zup and then click OK.

   The attributes for the Motion2 node should look like this:

   ![Motion2 Attributes](image)

6. Right-click the grid and choose Create Node, Blending, Blend Two.
7. Select the BlendTwo0 node.
   • In the Attributes pane, for Sync Mode, select Full Clip Based.
8. For Motion1 node, select the Output Pose box and drag the connector to the Pose 1 input of the BlendTwo0 node.
9. For **Motion2** node, select the **Output Pose** box and drag the connector to the **Pose 2** input of the **BlendTwo0** node.

10. For the **BlendTwo0** node, select the **Output Pose** box and drag the connector to the **Input Pose** of the **FinalNode0** node.

Your blend tree should look like the following:

![Blend Tree Diagram]

11. Right-click the grid and choose **Create Node**, **Sources**, **Parameters**.

12. Right-click the grid and choose **Create Node**, **Math**, **Smoothing**.

13. For **Parameters0** node, select the **speed** output box and drag the connector to the **Dest** input box of the **Smoothing0** node.

14. For the **Smoothing0** node, select the **Result** output box and drag the connector to the **Weight** input box of the **BlendTwo0** node.

Your blend tree should look like the following:

![Blend Tree Diagram]

15. In the **Animation Editor**, choose **File**, **Save All**. Then in the dialog box, click **OK**.

16. Navigate to the `/SamplesProject/AnimationSamples/Simple_JackLocomotion` directory. For the file name, enter `quickstart` and then click **Save** to save the workspace.

17. In the **Anim Graph** tab, click the **Play** button. The character should now be animated in the idle mode.

18. In the **Parameters** pane, move the **speed** slide control to the right to make Jack walk. Move the slider further to the right to make Jack run.
Referencing External Anim Graphs

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A node-based animation system that has thousands of nodes can be difficult to manage. Lumberyard's EMotion FX Animation Editor uses hierarchical nodes that help alleviate this problem, but universal-level changes to game logic can still be challenging.

Starting in Lumberyard version 1.18, you can use Animation Editor reference nodes to reference external animation graph (anim graph) files. This helps reduce the scale and complexity of anim graphs and minimize human error. Reference nodes behave as the root state machine of the anim graphs that they reference and always output one pose.

Topics
- Benefits (p. 1361)
- Using External Anim Graphs (p. 1361)
- Best Practices for Using Referenced Anim Graphs (p. 1371)
Benefits

Referencing anim graphs provide the following benefits:

- **Sharing anim graph pieces or snippets** – You can create anim graph pieces or snippets that can be shared in multiple other anim graphs. For example, you might build a locomotion anim graph part to share across all characters while individualizing the rest. When you use referencing, you don't need to copy and paste the same anim graph every time that you use it.

- **Ease of maintenance** – You can maintain a shared anim graph in one place. If you copy and paste anim graphs, each copy must be maintained separately.

  **Note**
  Because a change in a referenced anim graph can break the behavior of another, it is important to keep track of your referencing hierarchy. For more information, see Best Practices for Using Referenced Anim Graphs (p. 1371).

- **Greater ease of collaboration** – By clearly separating anim graphs, multiple people can develop animation for different characters simultaneously.

Using External Anim Graphs

This section shows you how to create reference nodes in the **Animation Editor**, assign external anim graphs to them, and view and manage referenced graphs.

**To create a reference to an external anim graph**

1. In the **Animation Editor**, do one of the following:
   - Right-click the **Anim Graph** grid and choose **Create Node, Sources, Reference**.
• Click the **Anim Graph Palette** tab. From **Sources**, drag and drop the **Reference** node to the grid.
The new reference node appears in purple in the Anim Graph grid.
2. Select the **Reference** node. The node color changes to orange when it is selected.
Referencing External Anim Graphs

The **Reference** section of the **Attributes** tab shows the attributes for the reference node.

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Specifies the name of the reference node.</td>
</tr>
<tr>
<td><strong>Anim graph</strong></td>
<td>Specifies the graph that is referenced. If none is provided, the reference node outputs the bind pose of the actor.</td>
</tr>
</tbody>
</table>
### Referencing External Anim Graphs

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Motion set asset</strong></td>
<td>Specifies the motion set (p. 1413) that the referenced anim graph uses. If a motion set is not specified, the referenced anim graph uses the motion set of the graph that contains the reference node. If a motion set asset is defined, an entry named Active motion set appears. Use this attribute to specify which motion set from the motion set asset to use. The root motion set from the motion set asset is selected by default.</td>
</tr>
</tbody>
</table>
| **Parameter mask**  | Specifies the anim graph parameters that are exposed as input ports on the reference node. When an anim graph is assigned to a reference node, the following occurs:  
  - The Animation Editor compares the names and types of parameters to detect which parameters can be mapped automatically.  
  - The parameters that are mapped successfully are not shown as input ports on the reference node.  
  - The parameters that can't be mapped automatically are added to the parameter mask, become inputs on the reference node, and use their default values.  
  You can change the mask to specify which parameters become input ports. Parameters that are not in the mask are subject to automatic mapping.  
  Even if a parameter is automatically mapped, you can remove it from the mask and manually specify it. This can be useful when the leader graph and referenced graph have parameters with the same name, but have a different range of possible values.  
  For example, a leader graph has a Speed value in the range \([0.0, 1.0]\). You want to reference a locomotion graph created by another team that has a Speed parameter in the range \([0.0, 100.0]\). To use the referenced graph, you can remap the Speed parameter for the leader graph to the correct input range for the referenced graph. You can use a visual connection to the graph to specify the correct value. |

3. On the Attributes tab, for Anim graph, click the (...) icon.
4. In the Pick EMotion FX Anim Graph dialog box, select the `.animgraph` file that you want to assign to the reference node, and then click OK.

5. In the Anim Graph grid, double-click the reference node to see the nodes that the referenced anim graph contains.
6. Continue double-clicking nodes to drill down into the nodes underneath. In this example, the referenced anim graph contains a `StateMachine` node, and the `StateMachine` node contains an `EntryNode` and an `ExitNode`.

When you view a referenced anim graph in this way, the referenced anim graph is read-only.
7. The display above the grid shows your current location in the node hierarchy. To go back to a previous node, click the node name.

8. On the upper right of the Anim Graph grid, click the navigation page icon.
The navigation pane opens up on the right side of the grid to show the hierarchy of nodes. The navigation pane displays all loaded anim graphs. The name of the current node is bold. The color of each node indicates its type. For example, the entry nodes are green and the exit nodes are red.

9. To edit an external anim graph, right-click the reference node that you assigned it to and choose Open 'filename.animgraph' file. The changes that you make to the external anim graph are reflected in all anim graphs that reference it.
Best Practices for Using Referenced Anim Graphs

See the following best practices for using referenced anim graphs:

- **Keep track of your referencing hierarchy** – One way to keep track of your referencing hierarchy is by maintaining a chart of your anim graphs and their references. Such a chart can help developers and testers know which anim graphs are affected by changes that are made. A chart can also help you know if an anim graph that you are working on is referenced by another anim graph.

- **Directory hierarchy** – Make your anim graph directory and file hierarchy the same as your referencing hierarchy. For example, anim graphs in directories higher up in the hierarchy use or reference anim graph assets deeper down the hierarchy, but not vice versa.

- **Minimize parameter count** – Keep the number of parameters in your referenced anim graphs minimal. Using many parameters increases complexity.

- **Manage motion sets effectively** – To manage motion sets when you use referencing anim graphs, consider the following options:
  - Manage separate motion sets. Each motion set contains the motions for one anim graph.
  - Create one large motion set for a leader anim graph. This motion set would hold motions for the leader anim graph and for all motions used in any of the referenced anim graphs.

  **Note**
  Both options allow the referenced anim graph to be tested by itself.

Tips for Working with Referenced Anim Graphs

Avoid the following practices when you work with referenced anim graphs:
• **Changing an anim graph that is referenced by another** – Changing an anim graph that is referenced by another anim graph can break its behavior. For example, if you remove a parameter from the referenced anim graph that another anim graph uses, the parameter reverts to its default value. This can cause unexpected behavior.

• **Renaming, moving, or deleting an anim graph** – When you rename, move, or delete an anim graph, its asset ID changes. Therefore, all anim graphs that refer to the renamed, moved, or deleted anim graph must also be updated. Having a system that keeps track of your referencing hierarchy (as mentioned in Best Practices (p. 1371)) makes it easy to know which anim graphs are affected and which to update.

### Synchronizing Animation Graphs: Example

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use synchronized animation graphs to synchronize animation between actors. For example, the animation of one actor might trigger an animation in another actor. An animation graph can be the primary graph and have multiple secondary graphs. Likewise, an animation graph can be both a secondary graph of one graph and a primary graph for another graph.

This topic describes the following main steps for synchronizing two animation graphs:

1. Add the required components to the entities to be synchronized, including the Anim Graph component.
2. Use Animation Editor to create a motion set and one or more animation graphs.
3. Add a parameter to the secondary graph to receive the change event from the primary graph.
4. Add a parameter to the primary graph.
5. Add a servant parameter action to the primary graph to send change events to the secondary graph.
6. Synchronize the graphs by using Lumberyard's Event Bus (EBus) (p. 1851) system and Lua script.

**Note**

For more information about synchronizing animation in separate clients across the network, see Synchronizing Animations Across a Network (p. 2051).

This topic illustrates this graph synchronization with an example that has two actors, a robot actor ("Jack") and a gun actor. When the player activates the sync mode and uses the keyboard to fire, the robot makes a firing motion and the gun fires. When the player deactivates the sync mode, the robot makes a firing motion, but the gun does not fire.
The robot has a syncFeature_Jack animation graph and the gun uses a syncFeature_Gun graph. The robot graph is the primary graph and the gun graph is the secondary graph.

When the sync mode is on, the gun fires in sync with the robot. When the robot fires, the state of the primary graph's shoot parameter is received by secondary graph's gunTrigger parameter. The secondary graph, which is attached to the gun, receives the parameter change event and fires the gun.
1. Add Required Components

The first step is to add the required components to the entities that you want to synchronize. The robot and gun components that are used by the entities in the example are described in the following sections.

Robot Entity Components

The robot entity uses the following components:

- Transform
- Actor
- Anim Graph
- Input
- Lua Script

The following image shows the configuration of the components for the robot entity in the Entity Inspector.
Gun Entity Components

The gun entity uses the following components:

- Transform
- Actor
- Lua Script
- Anim Graph
- Input

The following image shows the configuration of the components for the gun entity in the Entity Inspector.
2. Create a Motion Set and Animation Graphs

After you set up your entities and components, create a motion set and two animation graphs. The motion set contains the motions that your graphs use, and the secondary and primary animation graphs animate and synchronize the entities.

To create a motion set and two animation graphs

1. Create a motion set. For information about creating a motion set, see Getting Started with the Animation Editor (p. 1353). This example's MotionSet0 contains the motions gunshootanimation, jack_shoot, and jack_idle.

2. Create a secondary animation graph. For information about creating a secondary animation graph, see Getting Started with the Animation Editor (p. 1353). The secondary graph controls one or more entities whose actions are determined by the primary graph.

   This example's syncFeature_Gun secondary graph has a BindPose0 node and a Motion0 node. The Motion0 node contains the gunshootanimation motion.
3. Create a primary graph. The primary graph sends control events to the secondary graph.

This example's syncFeature_Jack primary animation graph has a **Motion1** node and a **Motion0** node. The **Motion1** node contains the jack_idle motion and the **Motion0** node contains the jack_shoot motion.
Synchronizing Animation Graphs

- Motion1
  - Name: Motion1
  - Probability weight: jack_idle 1.0
- Motion2
  - Name: Motion2
  - Probability weight: jack_choose 1.0
3. Add a Parameter to the Secondary Graph

You are now ready to add a parameter to the secondary graph that receives the parameter change event from the primary graph. After you add the parameter, add parameter conditions that specify when the animation transitions from one motion to another.

To add a parameter to the secondary graph to receive the change event from the primary graph

1. On the Parameters tab for the secondary animation graph, click the plus (+) icon, and choose Add parameter.

2. In the Create Parameter dialog box, for Value type, choose the data type that you want to use for the parameter. This example uses the Boolean (checkbox) value type because the gun trigger is either on or off.

3. For Name, enter a name for the parameter. This example uses gunTrigger.
4. Click Create. The Parameters list shows the parameter that you created.
Add Parameter Conditions to the Secondary Graph

In this section, add parameter conditions on the transition lines that specify when the animation changes. In the example, the conditions indicate whether the gun trigger has been pressed.

To add parameter conditions to the secondary graph

1. Click the transition line from the BindPose0 node to the Motion0 node. Then, in the Attributes pane, click Add condition, and choose Parameter Condition.

2. Click Select parameter.

3. In the Parameter Selection Window, choose the parameter that you just created, and click OK.
In the Attributes pane, a Parameter Condition section shows the parameter that you added. On the transition line, a small round node indicates that the line has a parameter condition.

4. For Test Function, use the default value of \( \text{param} > \text{testValue} \). In this example, this means that if the trigger receives a value greater than 0, the gun fires.

5. For Test Value, keep the default value of 0.0.

6. Click the transition line from the Motion0 node to the BindPose0 node. Then, in the Attributes pane, click Add condition, and choose Parameter Condition.
7. Click **Select parameter**.

8. In the **Parameter Selection Window**, choose the parameter that you are using. The example uses the `gunTrigger` parameter.

9. For **Test Function**, use the default value of `param == testValue`. In this example, this means that if the trigger value is equal to zero, the motion transitions back to idle, and the gun no longer fires.

10. For **Test Value**, keep the default value of 0.0.

Now the secondary animation graph is ready to receive signals from the primary graph.

### 4. Add a Parameter and Parameter Conditions to the Primary Graph

You add a parameter and parameter conditions to the primary graph just as you did with the secondary graph. However, you also add secondary ("servant") parameter actions to the primary graph. The actions signal the secondary graph to mimic the animations of the primary graph.

**To add a parameter to the primary graph**

1. On the **Parameters** tab, click the plus (+) icon, and choose **Add parameter**.
2. In the **Create Parameter** dialog box, for **Value type**, choose the data type that you want to use for the parameter. This example uses the **Boolean (checkbox)** value type because the gun trigger is either on or off.

3. For **Name**, enter a name for the parameter. This example uses `shoot`.
4. Click **Create**.

**Add Parameter Conditions to the Primary Graph**

Now you add parameter conditions on the transition lines in the primary graph as you did on the secondary graph.

**To add parameter conditions to the primary graph**

1. Click the transition line from the **Motion1** node to the **Motion0** node. Then, in the **Attributes** pane, click **Add condition**, and choose **Parameter Condition**.

2. Click **Select parameter**.
3. In the **Parameter Selection Window**, choose the parameter that you just created. This example uses `shoot`.

![Parameter Selection Window](image)

4. For **Test Function**, use the default value of `param > testValue`.
5. For **Test Value**, use the default value of `0.0`.
6. Click the transition line from the **Motion0** node to the **Motion1** node. Then, in the **Attributes** pane, click **Add condition**, and choose **Parameter Condition**.

![Add condition](image)

7. Click **Select parameter**.
8. In the **Parameter Selection Window**, choose the parameter that you are using. The example uses `shoot`.
9. For **Test Function**, use the default value of `param == testValue`.
10. For **Test Value**, use the default value of `0.0`.

### 5. Add Servant Parameter Actions to the Primary Graph

Now you are ready to add secondary ("servant") parameter actions to the primary graph. A script uses these actions to synchronize the two graphs.

**To add a servant parameter action to the primary graph**

1. In Animation Editor, click to select the first transition line again. This example selects the line from the **jack_idle** node **Motion1** to the **jack_shoot** node **Motion0**.
2. In the **Attributes** pane for the primary graph, click **Add action, Servant Parameter Action**.
3. For **Servant Parameter Action**, in the **Trigger Mode** box, keep the default **On Enter**. **On Enter** specifies that the action is triggered when the state or transition is entered.

4. In the **Servant anim graph** box, click **Browse (...)** and choose the secondary animation graph that has the parameter that you want to use. This example chooses a secondary animation graph called **syncFeature_Gun**.

5. Click **Select parameter** to choose a parameter from the secondary animation graph that you just chose. This example chooses the **gunTrigger** parameter.
On the transition line, a small square node indicates that the transition line has a parameter action. The small round node next to it represents the parameter condition that you added earlier.

6. For **Trigger value**, specify the value to emit when the action is triggered. Because **Trigger value** is treated as a single float, you can use it for float, Boolean, and integer parameters. This example specifies 1.0, which is the value when the gun fires.
7. Click the line from Motion0 to Motion1 and repeat the steps to add a servant parameter action to the remaining transition line.

8. In the Trigger value box, specify a different value. The example specifies a trigger value of 0.0, which is the value when the gun is idle.

Now that the animation graphs are ready, you can perform the next steps: gathering user input and writing Lua scripts to synchronize the graphs.

6. Synchronize the Primary and Secondary Graphs

Synchronizing the primary and secondary graphs involves the following steps:

1. Getting keyboard input from the player.
2. Placing a Lua script component and Lua script on the secondary entity.
3. Placing a Lua script component and Lua script on the primary entity.

The Lua scripts synchronize the two graphs by handling animation graph events in Lumberyard's Event Bus (EBus) (p. 1851) system.

Getting Input from the Player

In the example, the synchronization state of the primary and secondary graphs and the firing of the gun are controlled by the following keyboard inputs, or keystrokes, from the user. The Event Value Multiplier is the actual value sent to the input system and to Lua script.

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Description</th>
<th>Event Value Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Turns on sync mode.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Turns off sync mode (off by default).</td>
<td>-1</td>
</tr>
<tr>
<td>S</td>
<td>Fires the gun.</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>Stops the gun from firing.</td>
<td>-1</td>
</tr>
</tbody>
</table>

To gather these inputs, the example adds Input (p. 642) components to the robot entity and to the gun entity.

To use the Input component, you must enable the Input Management Framework (p. 1150) gem and the Starting Point Input (p. 1206) gem for your project. The Input Management Framework converts input to user-defined gameplay events. The Starting Point Input gem interprets hardware input and converts it into input events such as pressed, released, and held.

Each Input component references an .inputbindings file. An .inputbindings file binds a set of inputs to an event. These inputs can come from sources like a mouse, keyboard, or game controller.
You can use the **Input Bindings Editor** in Lumberyard Editor to create an input bindings file. For more information, see *Working with the Input Component (p. 643).*

**Getting Keyboard Input to Control Graph Synchronization**

In the example, the gun entity has an Input component. The Input component uses a `synctest.inputbindings` asset to bind keyboard inputs 1 and 2 to the `SyncControl` event. The `SyncControl` event controls the sync mode, which determines whether or not the gun fires when the robot fires.

The following image shows the corresponding input bindings in the **Input Bindings Editor**.

![Input Bindings Editor](image)

**Getting Keyboard Input to Control Shooting**

The Input component on the robot uses a `syncgun.inputbindings` asset to bind keyboard inputs S and D to the `ShootControl` event. The `ShootControl` event controls the firing of the gun.

The following image shows the corresponding input bindings in the **Input Bindings Editor**.
Using a Script on the Secondary Graph to Toggle Synchronization

The example uses Lua Script components on both the primary (robot) and secondary (gun) entities. The script on the primary entity controls the firing of the gun. The script on the secondary entity controls the sync mode. To add a Lua script to an entity, add a Lua Script component (p. 2684) to the entity and then attach the script to the component.

The Lua script on the gun entity receives the input from the keyboard to toggle the sync mode. To synchronize the secondary graph to the primary graph, the script uses the \texttt{SyncAnimGraph} EBus event. In the following example, the \texttt{self.entityId} parameter refers to the secondary entity (the gun). The \texttt{self.Properties.PrimaryEntity} parameter refers to the robot.

\begin{verbatim}
AnimGraphComponentRequestBus.Event.SyncAnimGraph(self.entityId,self.Properties.PrimaryEntity);
\end{verbatim}

To desynchronize the secondary graph from the primary graph, the same script uses the \texttt{DesyncAnimGraph} EBus event.

\begin{verbatim}
AnimGraphComponentRequestBus.Event.DesyncAnimGraph(self.entityId,self.Properties.PrimaryEntity);
\end{verbatim}

The full script shows how the \texttt{SyncAnimGraph} and \texttt{DesyncAnimGraph} methods handle the \texttt{SyncControl} input event.

\begin{verbatim}
-- syncSample.lua
local syncSample =
{
    Properties =
    {
        PrimaryEntity = { default = EntityId() }
    },
};
\end{verbatim}
function syncSample:OnActivate()
    self.SyncControlInputBusId = InputEventNotificationId("SyncControl");
    self.SyncControlInputBus = InputEventNotificationBus.Connect(self,
        self.SyncControlInputBusId);
    self.ShootControlInputBusId = InputEventNotificationId("ShootControl");
    self.ShootControlInputBus = InputEventNotificationBus.Connect(self,
        self.ShootControlInputBusId);
    self.SyncControl = false;
    self.Shooting = false;
end

function syncSample:HandleSyncControl(floatValue)
    if (floatValue > 0 and self.SyncControl == false) then
        AnimGraphComponentRequestBus.Event.SyncAnimGraph(self.entityId,
            self.Properties.PrimaryEntity);
        self.SyncControl = true;
    elseif (floatValue < 0 and self.SyncControl == true) then
        AnimGraphComponentRequestBus.Event.DesyncAnimGraph(self.entityId,
            self.Properties.PrimaryEntity);
        self.SyncControl = false;
    end
end

function syncSample:HandleInput(floatValue)
    if (InputEventNotificationBus.GetCurrentBusId() == self.SyncControlInputBusId) then
        self:HandleSyncControl(floatValue);
    end
end

function syncSample:OnPressed(floatValue)
    self:HandleInput(floatValue);
end

function syncSample:OnHeld(floatValue)
    self:HandleInput(floatValue);
end

return syncSample;

Using Script on the Primary Graph to Control Shooting

The example Lua script on the primary entity (the robot) receives the keyboard input that toggles
the firing of the gun. The robot entity's animation graph's shoot parameter uses the Boolean
(checkbox) type. When the gun fires, the shoot parameter is true. Because shoot is a named Boolean
parameter, the Lua script on the primary entity uses the SetNamedParameterBool function on the
AnimGraphComponentBus.

The full script shows how the SetNamedParameterBool function is used to toggle the shooting status.

-- syncGun.lua
local syncGun =

function syncGun:OnActivate()
    self.ShootControlInputBusId = InputEventNotificationId("ShootControl");
    self.ShootControlInputBus = InputEventNotificationBus.Connect(self,
        self.ShootControlInputBusId);
    self.SyncControl = false;
end
function syncGun:HandleShootControl(floatValue)
    if (floatValue > 0 and self.Shooting == false ) then
        AnimGraphComponentRequestBus.Event.SetNamedParameterBool(self.entityId, "shoot", true);
        self.Shooting = true;
    elseif(floatValue < 0 and self.Shooting == true ) then
        AnimGraphComponentRequestBus.Event.SetNamedParameterBool(self.entityId, "shoot", false);
        self.Shooting = false;
    end
end

function syncGun:HandleInput(floatValue)
    if (InputEventNotificationBus.GetCurrentBusId() == self.ShootControlInputBusId) then
        self:HandleShootControl(floatValue);
    end
end

function syncGun:OnPressed(floatValue)
    self:HandleInput(floatValue);
end

function syncGun:OnHeld(floatValue)
    self:HandleInput(floatValue);
end

return syncGun;

The Example in Action

The following animated image shows the finished example in action when the sync mode is turned off. The robot fires, but the gun does not.
The following animated image shows the example when the sync mode is turned on. The gun fires when the robot fires.
Animation Editor Components

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add the following Animation Editor components to an entity. You can then specify the actor file, animation graph file, the motion file, or attachment.

For more information, see the following components:

- Actor (p. 536)
- AnimGraph (p. 545)
• Attachment (p. 550)
• Simple Motion (p. 826)

For more information about adding a component to an entity, see Adding Components to an Entity (p. 479).

Using Morph Targets to Deform Characters

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A morph target is a deformed mesh that is stored as a series of vertex positions. Morph targets are also called blend shapes or vertex-level deformations. You can use morph targets to deform a character's face to animate facial expressions or a character's body part to correct undesired deformation of skinning. You can also simulate deformation of clothing on a character.

In the Animation Editor, you can use morph targets with one of the following nodes:
• Motion node – Plays morph target animations similarly as skeletal animations.
• Morph Target node – Animates morph targets by changing the weight at runtime.

Prerequisites

To use morph targets in the Animation Editor, you must do the following:
• Prepare your asset for .fbx export. For more information, see Customize FBX asset export with FBX Settings (p. 409).
• Create and animate morph targets on your character in your DCC tool (for example, Maya).
• Export the character as a .fbx file.

Topics
• Importing Morph Targets (p. 1399)
• Opening Actor Files (p. 1403)
• Previewing Morph Targets on Actors (p. 1404)
• Creating Motion Nodes with Morph Targets (p. 1405)
• Creating Morph Target Nodes (p. 1408)

Importing Morph Targets

When you import an .fbx file into Lumberyard, all morph targets and morph target motions in that file are imported as part of the actor. This allows you to open your actor file in the Animation Editor without additional steps.

You can also change how morph targets are imported.
To change how morph targets are imported

1. In the Asset Browser, right-click your .fbx file and choose Edit Settings.

2. In the FBX Settings window, click the Actors tab.

   A modifier appears to indicate that morph targets will be imported.

3. Click the button next to the Select morph targets field.
4. In the **Select nodes** window, select the morph targets that you want to import, and then click **Select**.
5. In the **FBX Settings** window, click **Update** to save your changes.

### To change how morph target motions are imported

1. In the **Asset Browser**, right-click your `.fbx` file and choose **Edit Settings**.
2. In the **FBX Settings** window, click the **Motions** tab.

**Note**

If you have morph target animations in your `.fbx` file, a modifier appears to indicate that morph target motions will be imported. You can remove the modifier if you do not want the morph target motions in your `.motion` file.
3. Click **Update** to save your changes.

### Opening Actor Files

When you open your actor file in the *Animation Editor*, all morph targets and morph target motions are imported by default. To change how morph targets are imported, see [Importing Morph Targets](#).  

**To open an actor file**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. In the *Animation Editor*, choose **File, Open Actor**.
3. In the *EMotion FX Actor* window, select an actor to import and then click **OK**.
Previewing Morph Targets on Actors

You can preview the morph targets on an actor.

To preview morph targets

1. In Lumberyard Editor, choose Tools, Animation Editor.
2. In the Animation Editor, choose View, Morph Targets.
3. In the Morph Targets window, preview morph target shapes on your actor by doing the following:
   a. Select the Select All check box.
      If enabled, the morph target sliders will override the morph target motions on your actor.
   b. Move the sliders next to the morph target name to see the actor's mesh deform.
Using Morph Targets to Deform Characters

c. Click **Edit** next to the morph target to adjust the range of the slider as needed. The default range is 0 to 1.

4. When you're done previewing the morph targets, clear the **Select All** check box and close the **Morph Targets** window.

**Creating Motion Nodes with Morph Targets**

Creating a motion node with morph targets is similar to other methods of motion node generation.

**To create a motion node with morph targets**

1. In the **Animation Editor**, on the **Motion Sets** tab, under **Motion Set Management**, do one of the following:
   
   • Click the + icon to create a motion set.
   
   • Click the folder icon to open the **Pick EMotion FX Motion Set** window and select a motion set to import. Click **OK**.
2. In the Anim Graphs pane, click the + icon to create an animation graph.

3. Drag the Motion node from the Sources tab in the Anim Graph Palette to the animation graph.
4. In the animation graph, select the **Motion** node that you added.

5. In the **Attributes** pane, click **Select motions**.
6. In the Motion Selection window, select the motion with the morph targets that you want to import and then click OK.

Creating Morph Target Nodes

As an alternative to creating a motion node with morph targets, you can use a Morph Target node to directly animate morph targets in the animation graph.

To create a morph target node

1. In the Animation Editor, drag the Blend Tree node from the Sources tab in the Anim Graph Palette to the animation graph.
2. In the animation graph, double-click the Blend Tree node. You should see a Final Node node and additional nodes in the Anim Graph Palette.
3. Drag the **Morph Target** node from the **Blending** tab in the **Anim Graph Palette** to the animation graph.

4. Do the following to add a bind pose to use as an input pose:
   a. Drag the **Bind Pose** node from the **Sources** tab in the **Anim Graph Palette** to the animation graph.
   b. Connect the **Output Pose** for the **Bind Pose** node to the **Input Pose** for the **Morph Target** node.

5. Do the following to add a parameter to control the weight of the morph target:
   a. Drag the **Parameters** node from the **Sources** tab in the **Anim Graph Palette** to the animation graph.
   b. In the **Parameters** pane, click the + icon to create a parameter.
   c. In the **Create Parameter** window, do the following:
      i. For **Value Type**, select **FloatSlider**.
      ii. For **Minimum** and **Maximum**, use the default values.
      iii. Click **Create**.
d. In the animation graph, do the following:

i. Connect the Parameter for the Parameter node to the Morph Weight for the Morph Target node.

ii. Connect the Output Pose for the Morph Target node to the Input Pose for the Final Node node.
6. In the animation graph, select the **Morph Target** node if it's not already selected.

7. In the **Attributes** pane, click **select morph targets**.

8. In the **Morph target selection** window, select the morph target that you want to import and then click **OK**.
The morph target is updated in the **Morph Target** node in the animation graph and in the **Attributes** pane.

9. In the **Anim Graphs** pane, activate the animation graph by double-clicking the name.

10. In the **Parameters** pane, move the slider to play the animation.
Customizing State Machine Routing with Sparse Motion Sets

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You might run into a case where you have different characters that don't have the exact same motions as the character for which you created the animation graph. Instead of duplicating large parts or creating new animation graphs for different characters, you can share the same animation graph for all your characters.

You can allow or deny transitions and routes to motion states in a state machine, based on the existence of the given motion entry.

Topics

- Sparse Motion Sets (p. 1413)
- Hierarchical Motion Sets and Overwriting Motion Entries (p. 1414)
- Customizing State Machines Based on Motion Sets (p. 1415)

Sparse Motion Sets

A motion set contains a list of motion entries. Each motion entry has a motion ID that you can define. For example, you can name a motion ID, Walk or Jump. The motion entry maps the custom motion ID like Walk to a specific motion file, such as /SampleProject/Animations/Human_Walk.motion.

You can add motion entries to a motion set with the + icon or the folder icon in the motion set window. The + icon creates a new motion entry where you can specify the motion ID, but does not assign it to
a motion file. You can click the folder icon to select a motion to add. Based on the motion file name, a
default motion ID is generated for you.

A *sparse motion set* is when the motion entries do not have a motion file assigned to them.

When you choose an unassigned motion entry on a motion state in a state machine, the motion shows
a red border around the node. If the motion state is activated, the character goes to a bind pose. In the
following example, the character's animation graph does not transition from idle to jump because the
*Jump* node is using a sparse motion set; no motion file is assigned. The character remains in the *Idle*
node.

### Hierarchical Motion Sets and Overwriting Motion Entries

A motion set can contain child motion sets that are stored in the same file. Child motion sets inherit all
motion entries from the parent.

In the following example, a hierarchical motion set has the *Human* motion at the root level. The *Zombie*
motion set is a child, which inherits from the parent *Human* motion set. The *Zombie* motion set does not
have a motion entry for *Jump*.

**Example : Hierarchical Motion Set**

**Human (motion set)**

- + Idle – GenericIdle.motion
- + Walk – GenericWalk.motion
- + Jump – GenericJump.motion

**Zombie (motion set)**

- + Walk – ZombieWalk.motion

Because the parent motion set defines the *Jump* motion entry with a motion file, the child motion set
inherits that motion. This means that when the *Zombie* motion set is activated, characters with that
motion set play the human *Jump* motion.

If you don't want your zombie characters to jump, you can disable inheritance for that specified motion.
In the child motion set, you can create a new motion entry named *Jump* and then mark it as unassigned.
This way you can override the motion entry from the parent, by not assigning it for the child motion set.

**Example : Child Motion Set with Unassigned Jump Motion**

**Human (motion set)**

- + Idle – GenericIdle.motion
Animation Editor Nodes

- + Walk – GenericWalk.motion
- + Jump – GenericJump.motion

Zombie (motion set)
- + Walk – ZombieWalk.motion
- + Jump – Unassigned

The Zombie motion set uses the Idle motion from the parent motion set, customizes the Walk motion, and disables the Jump motion.

For hierarchical motion sets, you can create a motion entry and unassign it to disable inheritance from the parent motion set. If you are not using hierarchical motion sets, this is the same as not having a motion entry with the specified motion ID.

Customizing State Machines Based on Motion Sets

Animation graphs can be shared across characters. Two different characters that use the same animation graph can operate with two different motion sets. For example, you can have a human character that uses the human motion set, and a zombie character that uses the Zombie motion set. Both characters can use the same animation graph.

You can configure the state machine to avoid motion states that are unassigned. In this example, you don't want the zombie to go into the Jump state, as this motion was unassigned for the zombie motion set.

To configure the state machine to avoid motion sets

1. In the Anim Graph, choose the transition line between your motion nodes. For example, you can select the transition line between the Idle and Jump nodes.
2. On the Attributes pane, click Add Condition and then select Motion Condition.
3. For Motion, select the Jump state.
4. For Test Function, select Is Motion Assigned?

Because the Zombie motion set does not have a motion file assigned for Jump, the character can't transition from the idle to jump state. The condition's traffic light appears red and blocks the transition. This lets you control whether a character is allowed to go to specific motion state or not.

Animation Editor Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
See the following nodes available in the Animation Editor.

**Topics**
- Creating Blend Trees (p. 1416)
- Controller Nodes (p. 1432)
- Simplify Node Groups with Hub Nodes (p. 1435)
- Using Math Nodes (p. 1438)
- Actions (p. 1443)

## Creating Blend Trees

This feature is in *preview* release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

In the **Animation Editor**, an animation graph consists of nodes and connections that define transitions and the way that data is passed between nodes. The **Animation Editor** supports state machines and blend trees. For more information about state machines, see About State Machines (p. 1348).

A blend tree is a collection of nodes with input and output ports that are color coded by data type and read from left to right. Input ports appear on the left side of the nodes and output ports appear on the right. The blend tree outputs the pose that is connected to the **Final Node**, which is included in every blend tree and cannot be deleted. If you choose not to connect a node to **Input Pose**, the **Final Node** outputs a bind pose.

### To create a blend tree

1. In Lumberyard Editor, choose **Tools**, **Animation Editor**.
2. In the **Animation Editor**, on the **Anim Graph** tab, click the + icon to create an animation graph.
3. Click the **Save** icon. Navigate to the directory where you want to save your animation graph. Type a name for your file and then click **Save**.
4. In the center pane, on the **Anim Graph** tab, right-click the grid and then choose **Create Node**, **Sources**, **Blend Tree**.
Alternatively, in the Anim Graph Palette, on the Sources tab, drag Blend Tree into the animation graph.

5. Double-click the blend tree node that you created. When you double-click the node, a new link appears above the animation graph with the node name. The Final Node also appears.
6. Do the following to add nodes and connections:
   a. In the animation graph, right-click the grid and choose Create Node. Choose a node from the following categories:
      - Sources
      - Blending
      - Controllers
      - Logic
      - Math
      - Misc
   b. Repeat step 6a to add more nodes to your blend tree.
   c. Connect the nodes by dragging inputs to outputs. Note the following color cues:
      - Gray – Dashed gray helper lines indicate the ports that you can connect to.
      - Green – The connection curve turns green when it's okay to release the mouse button.
      - Red – The connection curve turns red when the connection is not allowed.
      - Yellow – The connection curve turns yellow in transition states, such as when you drag a connection between ports.

Topics
- Creating and Visualizing Blend Spaces (p. 1419)
- Blending Poses with Blend Nodes (p. 1422)
Creating and Visualizing Blend Spaces

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Blend spaces are a collection of sample motions that are organized spatially according to their coordinates. The visual representation is a graph with xy-axes for the coordinates. The xy-axes can represent values such as move speed, travel direction, turn angle, and so on. Each motion is represented by a point (white dot) in the graph or blend space.

When you pick a point in the blend space (interactively or with parameter controls), your character automatically plays a resulting motion that is computed based on the sample motions and appropriate blend weights.

In a 1D blend space the motions correspond to points along a line. In a 2D blend space the motions correspond to points in a 2D space.

Prerequisites

Before you can add blend space nodes to the animation graph, you must have completed the following:

- Selected an actor
- Selected a motion set
- Created an animation graph

For more information, see Getting Started with the Animation Editor (p. 1353).

Creating Blend Spaces

To create a blend space, you must add a blend tree node and a blend space node, and then specify values for the attributes of the blend space node.

Blend space nodes output a pose from the Output Pose port. You can connect this output to the Input Pose port of the Final Output node or to the input port of any other node that accepts a pose as input.

The Blend Space 1D node has the following ports:

- X – The value for this input port indicates the current position of interest in the 1D blend space.
- Output Pose – The blend space node computes the blended motion that corresponds to the current position of interest and outputs the resulting motion from this port.

The Blend Space 2D node has the following ports:

- X – The value for this input port is the x-coordinate of the current position of interest in the 2D blend space.
- Y – The value for this input port is the y-coordinate of the current position of interest in the 2D blend space.
- Output Pose – The blend space node computes the blended motion that corresponds to the current position of interest and outputs the resulting motion from this port.
To create a blend space and specify attributes

1. In the Animation Editor, on the Anim Graph tab in the top middle pane, right-click the grid and then choose Create Node, Sources, Blend Tree.
2. Double-click the Blend Tree node to go to the blend tree view.
3. Add a blend space node to the blend tree by doing one of the following:
   - On the Anim Graph tab, in the blend tree view, right-click the grid and then choose Create Node, Blending, Blend Space 2D or Blend Space 1D.
   - In the Anim Graph Palette, on the Blending tab, drag the Blend Space 2D or Blend Space 1D icon into the blend tree view.
4. Double-click the blend space node to go to the blend space view. If you are using the Blend Space 2D node, your view should look as follows:

   ![Blend Space 2D view](image)

5. In the Attributes pane, specify values for the attributes of the blend space node. These values are used to set up your blend space.

   **Note**
   You can undock the Attributes pane in order to see the attributes and values without scrolling.
• To use provided values for the xy-axes, do the following:
  1. For Calculation method (X-Axis), select Automatically calculate motion coordinates.
  2. For X-Axis Evaluator, select a common motion characteristic.
  3. For Calculation method (Y-Axis), select Automatically calculate motion coordinates.
  4. For Y-Axis Evaluator, select another common motion characteristic.

• To use custom values for the xy-axes, do the following:
  1. For Calculation method (X-Axis), select Manually enter motion coordinates.
  2. For Calculation method (Y-Axis), select Manually enter motion coordinates.

You can also use a combination of provided and custom values. For example, you can manually enter motion coordinates for the x-axis and automatically calculate motion coordinates for the y-axis using the **Travel distance** evaluator.

6. In the **Attributes** pane, for **Motions**, click the + button to add the source motion assets for your blend space.

7. In the **Motion Selection Window**, choose the motions that you want to add to the blend space, and then click **OK**.

- The coordinate value automatically calculates if you selected **Automatically calculate motion coordinates** for Calculation method and if you selected a motion characteristic for the Evaluator.
- You must enter coordinate values if you selected **Manually enter motion coordinates** for Calculation method.

8. After your motions are added to the blend space and the coordinate values are calculated, verify that your blend space view looks similar to the following:
9. In the blend space view, do the following:
   a. Drag within the blend space to change the point of interest (represented by a red dot).
   b. When the point is highlighted, the corresponding motion is automatically computed by blending the motions represented by the three vertices of the triangle. View the blend weights next to each of the motions.
   c. Notice that the motions that are closer to the point have a higher blend weight than the motions that are farther away.
10. In the Attributes pane, you can do the following:
   - View the coordinate values for each motion
   - Change the values to remap the animations and the blend space graph
   - Remove a motion from the blend space

### Blending Poses with Blend Nodes

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Blend nodes in animation graphs to create an animation that blends two input poses.

**Animation Editor** has the following types of blend nodes, each of which blend poses in different manners:

- Blend N Node (p. 1423)
- Blend Two Additive Node (p. 1425)
- Pose Subtract Node (p. 1427)
- Blend Two Node (p. 1428)
- Blend Two (Legacy) Node (p. 1430)
Blend N Node

The Blend N node accepts up to ten inputs and uses the Weight parameter to determine which inputs to use and their weights. You can specify any type of parameter into the Weight input of a Blend N node.

To use the Blend N node

1. In Lumberyard Editor, choose Tools, Animation Editor.
2. Create a blend tree (p. 1416).
3. Double-click the blend tree node that you created.
4. Select the Anim Graph Palette tab and then select the Blending tab.
5. Drag the Blend N node into the animation graph.
6. Connect nodes to the following inputs and output:
• **Pose 0 to 9** – Pose inputs. Connect one or more inputs.
• **Weight** – Input that determines which pose inputs to use and their weights.
• **Output Pose** – Result of the blended poses.

7. Select the **Blend N** node.
8. For each pose, enter the **Max weight trigger** in ascending order.

**Example**

If you have three poses, you must specify values in ascending order. The first pose should have the lowest value and the last pose must have the highest.

If you enter a series of values in an invalid order, the value box turns red and displays a warning.
You can use the **Evenly Distribute** feature to automatically calculate an even distribution of weights.

**To distribute weights using the Evenly Distribute feature**

1. In the first input's **Max weight trigger**, enter the lowest value.
2. In the last input's **Max weight trigger**, enter the highest value.
3. Click **Evenly Distribute**. This function calculates and evenly spaces the values.

**Example**

You have four inputs. The lowest input is set to 0.0 and the highest input is set to 1.0. Once you click **Evenly Distribute**, your middle values are automatically calculated to be spaced evenly between 0 and 1. The final values would be 0.0, 0.33, 0.66, and 1.0.

The value of the **Weight** parameter determines which inputs to blend by its value with respect to the **Max weight trigger** values. The **Weight** value naturally falls either before the lowest **Max weight trigger** values, between two values, or after the highest **Max weight trigger** value. If it's lower than the lowest **Max weight trigger** value, then only that pose is used in the calculation. If it's higher than the highest **Max weight trigger** value, then only that pose is used. If it's between two values, then those two poses are used.

**Example**

Input ports **Pose 5**, **Pose 7**, and **Pose 9** are connected, with **Max weight trigger** values of -2.0, 4.0, and 8.0. If the input value is less than or equal to -2.0, then only the port **Pose 5** is used to calculate the output pose. If the input is between -2.0 and 4.0, both ports **Pose 5** and port **Pose 7** are used to calculate the output pose. If the weight is greater than 8.0, then only port **Pose 9** is used.

The weight assigned to each value in a pair depends on where the **Weight** parameter falls with respect to the pose values. It calculates the respective distances and assigns a weight based on its position.

**Example**

The **Weight** input is set to 0.0. **Pose 5** is set to -1.0, and **Pose 7** is set to 3.0, which is a difference of 4.0. Because the value of 0.0 is at the 25% point between -1.0 and 3.0, a weight of 0.25 is assigned to **Pose 5**. The remainder (0.75) is assigned to **Pose 7**.

**Blend Two Additive Node**

The **Blend Two Additive** node blends a pose 2 input additively on top of a pose 1 input.

**Note**

This node functions similarly to the **Blend Two (Legacy)** node with **Additive** enabled. The key difference is that the **Blend Two (Legacy)** node subtracts the bind pose when applying an
additive blend. Furthermore, **Blend Two Additive** expects pose 2 to be an additive pose. This means that if you wanted **Blend Two Additive** to function like the **Blend Two (Legacy)** node, you must first subtract the bind pose from pose 2.

**To use the Blend Two Additive node**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. Create a blend tree (p. 1416).
3. Double-click the blend tree node that you created.
4. Select the **Anim Graph Palette** tab and then select the **Blending** tab.
5. Drag the **Blend Two Additive** node into the animation graph.

6. Connect nodes to the following inputs and output:
   - **Pose 1** – The base pose.
   - **Pose 2** – The pose to be added to **Pose 1**.
   - **Weight** – Weight of additive.

   You can use a **Float Constant** node, for example, to specify a float value between 0.0 and 1.0. A value of 0.0 means **Pose 2** does not affect **Pose 1** at all. A value of 1.0 means that **Pose 2** is fully added on top of **Pose 1**. Other nodes that can specify **Weight** include the **Parameter** node, the **Smoothing** node, and so on.
   - **Output Pose** – The result of the blended poses, which you can visualize as **Pose 1 + (Pose 2 * Weight)**.

**Blend Two Additive Node Attributes**

For attribute settings that are shared among the blend node types, see **Blend Node Attributes** (p. 1430).

The **Extraction Mode** for the **Blend Two Additive** node features masking and additive blending, which adds complexity over **Extraction Mode** for transitions.
The output from motion extraction with the **Blend Two Additive** node is calculated as follows.

- **S** = Source transform delta
- **T** = Target transform delta

**Additive, root included in mask (or no mask provided):**

- **Blend** = **S** + **T** * weight
- **Source** = **S**
- **Target** = **T**

**Additive, root excluded from mask:**

- **Blend** = **S**
- **Source** = **S**
- **Target** = **S**

**Pose Subtract Node**

The **Pose Subtract** node subtracts **Pose 2** from **Pose 1**. The output of a **Pose Subtract** node is the delta between them (**Pose 1** – **Pose 2**).

**Note**

With the **Pose Subtract** node, you can generate an additive pose. You can then supply it to the **Blend Two Additive** node at runtime without manually generating it from a DCC. The output of the **Pose Subtract** node cannot be used as an input to the **Blend Two (Legacy)** node. This is because **Blend Two (Legacy)** does not expect a pose that is already a delta or additive pose, but expects a bind pose.

**To use the Pose Subtract node**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. Create a **blend tree** (p. 1416).
3. Double-click the blend tree node that you created.
4. Select the **Anim Graph Palette** tab and then select the **Blending** tab.
5. Drag the **Pose Subtract** node into the animation graph.
6. Connect nodes to the following inputs and output:

- **Pose 1** – The base pose.
- **Pose 2** – The pose to be subtracted from the base pose.
- **Output Pose** – The result of the pose subtraction (Pose 1 - Pose 2).

**Pose Subtract Node Attributes**

For attribute settings that are shared among the blend node types, see [Blend Node Attributes](p. 1430).

**Blend Two Node**

With the **Blend Two** node, you can blend between two input poses based on a weight value. For example, the **Blend Two** node can blend smoothly between a walk and a run based on the character's speed.

The **Blend Two** is similar to the **Blend Two Additive** node, except that it doesn't support **Additive Blend** mode.

**To use the Blend Two node**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. Create a blend tree (p. 1416).
3. Double-click the blend tree node that you created.
4. Select the **Anim Graph Palette** tab and then select the **Blending** tab.
5. Drag the **Blend Two** node into the animation graph.
Connect nodes to the following inputs and output:

- **Pose 1** – The first pose.
- **Pose 2** – The second pose.
- **Weight** – Blend weight.

You can use a Float Constant node, for example, to specify a float value between 0.0 and 1.0. A value of 0.0 means 100% of Pose 1 and 0% of Pose 2. A value of 0.6 weights 40% of Pose 1 and 60% of Pose 2. Other nodes that can specify Weight include the Parameter node, the Smoothing node, and so on.

- **Output Pose** – The result of the pose blending.

**Blend Two Node Attributes**

For attribute settings that are shared among the blend node types, see Blend Node Attributes (p. 1430).

The **Extraction Mode** for the **Blend Two** node is calculated as follows.

- **S** = Source transform delta
- **T** = Target transform delta

**Root included in mask (or no mask provided):**

- **Blend** = S + (T - S) * weight
- **Source** = S
- **Target** = T

**Additive, root excluded from mask:**

Version 1.28
1429
- **Blend** = S
- **Source** = S
- **Target** = Zero

**Blend Two (Legacy) Node**

The **Blend Two (Legacy)** supports backward compatibility. If you are creating new content, use the **Blend Two** (p. 1428) or **Blend Two Additive** (p. 1425) node.

**Blend Node Attributes**

The blend nodes feature a set of attributes that control different aspects of how the two nodes are blended. Some blend nodes have different attributes, which are described in the sections about that node type.

**Sync Mode**

The **Sync Mode** attribute determines the method of synchronizing motion clips to keep the feet synchronized.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>Synchronization is disabled.</td>
</tr>
<tr>
<td>Event Track Based</td>
<td>Synchronization based on sync event tracks. This method pairs sync events,</td>
</tr>
<tr>
<td></td>
<td>which ensures that the events in each of the inputs activate at the same</td>
</tr>
<tr>
<td></td>
<td>time.</td>
</tr>
<tr>
<td>Full Clip Based</td>
<td>Synchronization based on the full clip duration. Inputs are synchronized</td>
</tr>
<tr>
<td></td>
<td>by the percentage that is complete. For example, when the first input is at</td>
</tr>
<tr>
<td></td>
<td>25% playback of the input's duration, the second input is also at 25%</td>
</tr>
<tr>
<td></td>
<td>playback.</td>
</tr>
</tbody>
</table>

**Event Filter Mode**

The **Event Filter Mode** attribute determines which node's events are emitted.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader Node Only</td>
<td>Emits events from the leader node only. The follower node is synchronized to</td>
</tr>
<tr>
<td></td>
<td>this node.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>Follower Node Only</td>
<td>Emits events from the follower node only. The follower node is synchronized to the leader node.</td>
</tr>
<tr>
<td>Both Nodes</td>
<td>Emits events from both leader and follower nodes.</td>
</tr>
<tr>
<td>Most Active</td>
<td>Emits events from the more active node. Special use case for additive blends.</td>
</tr>
</tbody>
</table>

**Extraction Mode**

The **Extraction Mode** attribute controls how the motion extraction behaves when blending. For example, for transitions inside state machines, you can use this node to ensure that a 180 degree turn completes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Blend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blend</td>
<td>Blends between the source and target. This setting helps ensure a proper blending result. For example, when transitioning from idle into a turn, translation and rotation are blended. Some of the turn's rotation can get lost in the blending, which means a 180 degree turn might reach only 160 degrees. Use this blend setting to ensure the completion of the turn.</td>
</tr>
<tr>
<td>Target Only</td>
<td>Extracts the target only. For example, when transitioning from idle into a turn, only the turn animation's translation and rotation is extracted. The nodes inside the skeleton still blend normally. This affects only the character's rotation and translation.</td>
</tr>
<tr>
<td>Source Only</td>
<td>For example, when transitioning from idle into a turn, only the idle pose's translation and rotation is extracted. The nodes inside the skeleton still blend normally. This affects only the character's rotation and translation.</td>
</tr>
</tbody>
</table>
Mask

Use **Mask** to select the skeleton nodes to include in the blend.

Example

A walking motion might be your first input pose and a waving motion your second input pose. If you select as a mask the arm, hand, and fingers, and increase the pose’s weight, the arm would wave while the body walks. In this example, it would blend the arm bones from walk to wave.

Controller Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/) or visit the AWS Game Tech blog to learn more.

See the following controller nodes in the **Animation Editor**.

**Topics**

- Get Transform (p. 1432)
- Set Transform (p. 1434)

**Get Transform**

The **Get Transform** node gets the transform data from a joint that you specify. The transform includes the translation (position), rotation, and scale. You can use this node to return the transform for your animations.

**To create a Get Transform node**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. In the **Animation Editor**, on the **Anim Graph** tab, open an existing animation graph or click the + icon to create one.
3. Right-click the graph and choose **Create Node, Sources, Blend Tree**.
4. Double-click the **Blend Tree** node, right-click the graph, and then choose **Create Node, Controllers, Get Transform**.

![Create Node and Get Transform](image)

The **Get Transform** node appears in your graph.
5. To get the transform from a specific animation, right-click the graph and choose Create Node, Sources, Motion.

6. Connect the Output Pose of the Motion node to the Input Pose of the Get Transform node.

7. For the Get Transform node, in the Attributes tab, click Select node. This selects the joint from which you want to get the transform.

8. In the Node Selection Window, select your preferred joint and click OK.
The **Get Transform** node will output the vector \((x, y, z)\) for **Output Translation** (position), **Output Rotation**, and **Output Scale**.

9. In the right pane, on the **Attributes** tab, specify the **Transform Space**. You can specify the following.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>The transform relative to the parent.</td>
</tr>
<tr>
<td>World</td>
<td>The transform relative to the world.</td>
</tr>
<tr>
<td>Model</td>
<td>The transform relative to the model's origin.</td>
</tr>
</tbody>
</table>

**Set Transform**

The **Set Transform** node sets the transform from a selected joint, including translation (position), rotation, and scale.

**To create a Set Transform node**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. In the **Animation Editor**, on the **Anim Graph** tab, open an existing animation graph or click the + icon to create one.
3. Right-click the graph and choose **Create Node, Sources, Blend Tree**.
4. Double-click the **Blend Tree** node, right-click the graph, and then choose **Create Node, Controllers, Set Transform**.

Version 1.28

1434
The **Set Transform** node takes **Input Pose**, **Translation** (position), **Rotation**, and **Scale** as inputs, and then outputs **Output Pose** for selected nodes (joints).

5. For the **Set Transform** node, on the **Attributes** tab, click **Select joint**. This specifies the joint from which you want to set the transform.

6. Select your preferred joint and click **OK**. The **Set Transform** node will output the vector \((x, y, z)\) for **Output Pose**.

7. In the right pane, on the **Attributes** tab, specify the **Transform space**. You can specify the following.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>The transform relative to the parent.</td>
</tr>
<tr>
<td>World</td>
<td>The transform relative to the world.</td>
</tr>
<tr>
<td>Model</td>
<td>The transform relative to the model's origin.</td>
</tr>
</tbody>
</table>

### Simplify Node Groups with Hub Nodes

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/training/tutorials/open-3d-engine/) or visit the [AWS Game Tech blog](https://aws.amazon.com/blog/game/) to learn more.

The **Hub** node acts as a connection point between groups of nodes in an animation graph. This pass-through node outputs or forwards the pose of the node that entered it. Hub nodes reduce the complexity of transitions by acting as the central point in a state machine. By connecting multiple nodes to hubs, you can combine transition lines that share the same transition conditions and strategically organize nodes to create an easy-to-read state machine.

**Example**

The following animation graph has many groups and transitions.
The same graph with **Hub** nodes simplifies transitions between the groups, which makes the graph cleaner and easier to read.

**To use Hub nodes in your animation graph**

1. Add a **Hub** node in an animation graph by doing one of the following:
   - In the **Anim Graph Palette**, choose the **Sources** tab and drag **Hub** to the graph.
   - In the graph, right-click and choose **Create Node, Sources, Hub**.
2. Repeat to add as many **Hub** nodes as you need.

3. Add multiple like nodes, such as motion nodes or state machines, between the **Hub** nodes and create transitions.

**Example**

Transitions between motions are simplified by placing them between two **Hub** nodes.

In the example, the transition between **attack01** and **Hub0** is a shared transition. If you remove the **Hub0** node, you would then individually add those two conditions from **attack01** to each of the four transitions going into the **attack01** nodes.

4. For all transitions entering a **Hub** node, set its **Transition Time** to 0.0 seconds. This ensures that an extra delay is not added to the transition.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard's Animation Editor features a set of math nodes that perform math operations on various types of input. The math nodes pass on the result(s) of the operation as output.

**Topics**

- Rotation Math 2 Node (p. 1438)
- Rotation Limit Node (p. 1440)
- Vector Decompose Nodes (p. 1441)
- Boolean Logic Node (p. 1442)

**Rotation Math 2 Node**

With the Rotation Math 2 node, you can apply math operations to input rotations, which are represented by quaternions (p. 3285).

This node multiplies the input rotation and the specified Default Value to express an output rotation. The Default Value specifies the rotation of an unconnected input. This rotation value is expressed in Euler angle degrees (rotation about the X, Y, and Z axes).
To use the Rotation Math 2 node

1. Connect the **Output Rotation** output rotation output(s) to the input(s) of the **RotationMath2** node.
2. Select the **RotationMath2** node.
3. In the right pane, on the **Attributes** tab, specify the **Math Function**. You can specify the following:
   - **Rotate**
     
     Multiplies two input quaternions or one input with the **Default Value**.
   - **Inverse rotate**
     
     Multiplies the x input with the inverse of the y input. You can also use this to calculate the relative rotation of X with respect to Y.
4. If only one input rotation exists, specify the unconnected rotation values (X, Y, Z) for the **Default Value**.
Rotation Limit Node

With the Rotation Limit node, you can limit an input rotation. To do this, the node decomposes the rotations of the quaternion along relevant axes and limits their angles to a range. You define the minimum and the maximum possible angle values to remove ambiguity between the shortest or longest path angle defined by two values.

To use the Rotation Limit node

1. Connect a rotation output to the Input Rotation of the RotationLimit node.
2. Select the RotationLimit node.

3. In the right pane, on the Attributes tab for Rotation limits, enter Min angle and Max angle values for X, Y, and Z.

   **Note**
   
   The Min angle can't exceed the Max angle. If it does, the error is displayed in red and the value doesn't commit to the graph.

4. For Twist axis, select the X, Y, or Z Axis.

   The Twist axis specifies which axis the Rotation Limit node is to decompose and apply edited constraints.

---

**Vector Decompose Nodes**

Using Vector Decompose nodes, you can output one or more specific values of a vector.

**Example**

You have a 3D vector XYZ that indicates a position in a 3D world, but you only need its height (Z) for a computation. You would input your vector into a Vector3Decompose node and use only the Z output for your calculation.

You don't need to use Vector Decompose nodes if you are simply adding or subtracting one of the X, Y, Z, or W positions. For typical Vector3 to Vector2 (and vice versa) conversion, or Vector3 to Vector4 (and vice versa) conversion, the Animation Editor automatically converts vectors in the following way:

- Vector2 to Vector3 – Adds the Z component set to 0.
• **Vector3** to **Vector2** – Ignores the Z component from **Vector3**.
• **Vector3** to **Vector4** – Adds the **W** component set to 0.
• **Vector4** to **Vector3** – Ignores the **W** component from **Vector4**.

### Boolean Logic Node

Using the **Boolean Logic** node, you can apply a function to two boolean inputs. Boolean values are always 1 or 0 (true or false), such as a check box item. The **Boolean Logic** node sees any non-zero value as true (1) and any zero value as false (0). For example, values 0.54, 10.43 or -2.25 are all true (1). Only 0.0 values are false (0).

When choosing an output type, you can output from the **Float** output or the **Bool** output. The **Bool** output passes on a 0 or 1. The **Float** output passes on a float value that you specify in the attributes.

### Boolean Logic Node Attributes

The **Boolean Logic** node features a set of attributes that performs operations on boolean values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Name of the node.</td>
</tr>
<tr>
<td><strong>Logic Function</strong></td>
<td>You can set the following functions on boolean inputs:</td>
</tr>
<tr>
<td></td>
<td>• <strong>AND</strong> – Outputs true (1) if both x and y are true (1).</td>
</tr>
</tbody>
</table>
## Animation Editor Nodes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>Outputs true (1) if either x or y are true (1).</td>
</tr>
<tr>
<td>XOR</td>
<td>Means &quot;exclusive or.&quot; Outputs true (1) if x and y have different values. Otherwise, false (0).</td>
</tr>
<tr>
<td>NAND</td>
<td>Means &quot;not and.&quot; Outputs false (0) if both x and y are true (1). Otherwise, true (1).</td>
</tr>
<tr>
<td>NOR</td>
<td>Means &quot;not or.&quot; Outputs true (1) if both x and y are false (0).</td>
</tr>
<tr>
<td>XNOR</td>
<td>Means &quot;exclusive not or.&quot; Outputs true (1) if x and y have the same value. For example, if x and y are true or if x and y are false, outputs true (1).</td>
</tr>
<tr>
<td>Not x</td>
<td>Outputs true (1) if x is false (0).</td>
</tr>
<tr>
<td>Not y</td>
<td>Outputs true (1) if y is false (0).</td>
</tr>
</tbody>
</table>

**Default Value**

Used as a second value when only one input value is specified.

**Float Result When True**

Sets a float value to output when the result of the boolean function is true (1). You must also use the Float output connector to output this value.

**Float Result When False**

Sets a float value to output when the result of the boolean function is false (0). You must also use the Float output connector to output this value.

---

**Actions**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use actions to trigger parameter value changes when an animation graph state or transition reaches a specified state.

You can set the Parameter Action on the following:
State

Set actions to trigger upon:
- Entering a state – Action triggers when the state is fully blended into the node and is no longer in transition
- Exiting a state – Action triggers when the state is fully blended out of the node and the node is no longer active

Transition

Set actions to trigger upon:
- Entering a transition – Action triggers immediately upon starting a transition
- Exiting a state – Action triggers when a transition is fully blended to the target state

Adding Actions to a State

Add an action to a state to achieve a parameter value change that triggers after the state is either fully blended into the node or fully blended out of the node.

To add an action to a state

1. In the Animation Editor, open or create an animation graph.
2. Select a node.
3. In the Attributes panel, click Add action and then Parameter Action.
4. In the Attributes panel, under Parameter Action, select a Trigger Mode. You can select one of the following options:
   - On Enter – Executes a State action when the state is fully blended into the node and is no longer in transition
   - On Exit – Executes a State action when the state is fully blended out of the node and the node is no longer active
5. Choose Select parameter and select your preferred parameter.
   The selected parameter name replaces the text in the Select parameter box.
6. To turn the action on, set the Trigger Value to 1.

Adding Actions to a Transition

Add an action to a transition to achieve a parameter value change that triggers either when the transition starts or when the transition is fully blended to the target state.
To add an action to a transition

1. In the Animation Editor, open or create an animation graph.
2. Select a transition.

3. In the Attributes panel, click Add action and then Parameter Action.

The following options trigger the change on a follower graph but with different inputs to the follower graph's parameter:

- **Follower Parameter Action** – A constant value is provided
- **Symbolic Follower Parameter Action** – A user-selected value is provided

4. In the Attributes panel, under Parameter Action, select a Trigger Mode. You can select one of the following options:

- **On Enter** – Executes the action immediately upon starting a transition
- **On Exit** – Executes the action when a transition is fully blended to the target state
5. Choose **Select parameter** and select your preferred parameter.

   The selected parameter name replaces the text in the **Select parameter** box.

   ![Select parameter and Aim](image)

6. To turn the action on, set the **Trigger Value** to 1.

   **Note**
   When you set a trigger on a transition, a square appears on the transition line, which represents the action.

---

**Using Tags with Animation Graphs**

This feature is in **preview** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the [AWS Game Tech blog](https://aws.amazon.com/blogs/gaming/) to learn more.

In the **Animation Editor**, you use tags to describe the current state of your character and control the transition between different states. Tags are Boolean flags that are either active (enabled) or inactive (disabled). Some examples of tags are Happy, Holding Sword, and Left Leg Injured.

**Adding Tags**

Tags are represented by animation graph parameters. When you define a parameter, you can specify a different value for each entity that uses the same animation graph and parameter. For example, you
can specify a different value for the Speed parameter for each entity that uses the animation graph. Similarly, you can assign a different tag to each entity. For example, one entity has the Holding Sword tag active and another entity has the Happy tag active. For more information about parameters, see About Parameters (p. 1345).

To create a tag

1. In Lumberyard Editor, choose Tools, Animation Editor.
2. In the Animation Editor, in the Parameters pane, click the + button.

   ![Parameters Pane]

3. In the Create Parameter dialog box, do the following:
   a. For Value type, select Tag.
   b. For Name, enter a name for your tag.
   c. For Description, enter an optional description for your tag.
   d. For Default, select the check box to enable the tag.
4. Click **Create**.

### Adding Conditions to Tags

Use tag conditions to enable the state machine to change the active state. For example, you can choose a specific jump animation based on the active tag. To transition to Awesome Jump, you would enable the Freaky, Awesome, and Happy tags.
You can also use tags in combination with wildcard transitions to choose a specific state that other states can access. Wildcard transitions are transitions that can originate from any node. In the preceding example, the arrow to the left of `Idle` represents the wildcard transition. This means you can transition from any state to the `Idle` state, as long as the condition for the wildcard transition is met.

Tag conditions have two attributes: test function and tags.

**Test Function**

Specifies the tag status to pass the condition.

You can choose from the following options:

- **All tags active** – All tags must be active or the condition blocks the change.
- **One or more tags inactive** – At least one tag must be inactive or the condition blocks the change.
- **One or more tags active** – At least one tag must be active or the condition blocks the change.
- **No tag active** – All tags must be inactive or the condition blocks the change.

**Tags**

Specifies the tags that the condition checks for.

To add a tag to a condition, select the transition line between your nodes. In the Attributes pane, select the values that you want to use.

**Note**

You can only choose from tags that are available in the Parameters pane. For more information, see Adding Tags (p. 1446).

**Customizing EMotion FX Objects**

This feature is in preview release and is subject to change.
The EMotion FX API supports registering custom object types, including state machine nodes, blend tree nodes, transitions, and conditions. You can define custom object types in your game code or a custom gem. This allows you to have granular control of the Lumberyard animation system.

Registering Custom Objects

Before registering custom objects, activate the EMotion FX SystemComponent to ensure the EMotion FX runtime is initialized correctly. Then use an EBus call to the EMotionFXRequestBus::Events::RegisterAnimGraphObjectType method. You can ensure that EMotion FX runtime is activated by registering your custom node from a component that has a dependency on EmotionFXAnimationService. You do not need to manually instantiate the EMotion FX SystemComponent and call Activate; component dependencies handle these tasks.

To register your custom node

1. In your custom gem or game project code, define your subclass of EMotionFX::AnimGraphObject.
2. Create a subclass of AZ::Component.
3. In your component's GetDependentServices() method, add the dependency to EmotionFXAnimationService:

   ```cpp
   dependent.push_back(AZ_CRC("EmotionFXAnimationService", 0x3f8a6369));
   ```

4. In your component's Activate() method, register your node type:

   ```cpp
   EmotionFXAnimation::EMotionFXRequestBus::Broadcast(
       &EmotionFXAnimation::EMotionFXRequestBus::Events::RegisterAnimGraphNodeType,
       MyCustomNode::Create(nullptr)
   );
   ```

Implementing AnimGraphObject Subclasses

AnimGraphObject is the base class for all objects in the animation graph. The constructor on the base class is protected; instead, objects are instantiated with the Create() method. The Lumberyard animation system (EMotion FX) uses an instance of AnimGraphObject to create other instances by calling the Clone() method.

Each AnimGraphObject subclass has a unique type ID that is used to serialize an object to and unserialize an object from an .animgraph file. You use a public anonymous enum with a TYPE_ID member to declare the type ID for an object.

When implementing an AnimGraphObject subclass, you must define the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32 GetBaseType() const</td>
<td>Defines the base type of an object. There are three base types: nodes, transitions, and conditions.</td>
</tr>
<tr>
<td>const char* GetTypeString() const</td>
<td>Defines the string version of the object type name.</td>
</tr>
</tbody>
</table>
Creating Custom Motion Events and Parameters Using C++

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can author custom event classes with custom parameters and reflect them to the edit context for use in Animation Editor. By creating predefined motion events with known parameters, you minimize the performance impact of complex parameters and reduce the risk of human error.

In Animation Editor, users can perform the following tasks with the events and parameters that you create:

- Choose the events to add to motions
- Provide their own values for the parameters
- Add additional parameters to an event instead of creating duplicate events when the timing for two events is the same
- Quickly create events with complex parameters
- Tag motion events with colors that they choose and create a color scheme that is meaningful for them
- Save the event and parameter information in an XML file that can be reloaded

Topics
- Creating Motion Events (p. 1451)
- Creating EventData Types (p. 1455)

Creating Motion Events

The EMotionFX::MotionEvent class inherits EMotionFX::Event and describes an event that happens at a specified time or range of time during a motion.

A motion event could be a footstep sound to play, a particle system to spawn, or a script to execute. Because motion events are completely generic, EMotion FX doesn't handle them for you. You must create and handle the events that your game requires.
Each motion event has a list of `EventData` instances that are attached to the event. An event handler uses the `EventData` list to perform the required actions.

All motion events are stored in a motion event table. The motion event table (see `Motion.h`) contains data for the event types and parameters that can be shared between events.

To listen to motion events, connect to the `ActorNotificationBus` and implement `OnMotionEvent()`.

For information on creating `EventData` instances and adding motion events to a motion, see Creating `EventData` Types (p. 1455).

Topics
- MotionEvent Public Member Functions (p. 1452)

MotionEvent Public Member Functions

The `MotionEvent` class includes the following public member functions.

Functions
- `MotionEvent` (p. 1452)
- `SetStartTime` (p. 1453)
- `SetEndTime` (p. 1453)
- `GetStartTime` (p. 1454)
- `GetEndTime` (p. 1454)
- `GetIsTickEvent` (p. 1454)
- `ConvertToTickEvent` (p. 1454)
- `GetIsSyncEvent` (p. 1454)
- `SetIsSyncEvent` (p. 1454)
- `HashForSyncing` (p. 1454)

**MotionEvent**

You can use the `MotionEvent` function to trigger an event at a specific point in time (a tick event) or during a specified time range (a range event). To specify the data that the event emits, you can use pointers or datasets.

**Syntax**

```
MotionEvent (float timeValue, EventDataPtr &&data)
```

Creates a tick event and uses a data pointer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeValue</td>
<td>The time value, in seconds, when the motion event should occur.</td>
</tr>
<tr>
<td>data</td>
<td>The values to emit when the event is triggered.</td>
</tr>
</tbody>
</table>

**Syntax**

```
MotionEvent (float startTimeValue, float endTimeValue, EventDataPtr &&data)
```
Creating Custom Motion Events and Parameters Using C++

Creates a ranged event and uses a data pointer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTimeValue</td>
<td>The start time value, in seconds, when the motion event should start.</td>
</tr>
<tr>
<td>endTimeValue</td>
<td>The end time value, in seconds, when the motion event should end. When equal to the start time value, a start event is triggered, but no end event occurs.</td>
</tr>
<tr>
<td>data</td>
<td>The values to emit when the event is triggered.</td>
</tr>
</tbody>
</table>

**Syntax**

```
MotionEvent (float timeValue, EventDataSet &&datas)
```

Creates a tick event and uses a dataset.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeValue</td>
<td>The time value, in seconds, when the motion event should occur.</td>
</tr>
<tr>
<td>datas</td>
<td>The values to emit when the event is triggered.</td>
</tr>
</tbody>
</table>

**Syntax**

```
MotionEvent (float startTimeValue, float endTimeValue, EventDataSet &&datas)
```

SetStartTime

Sets the start time value of the event, which is when the event should be processed.

**Syntax**

```
void SetStartTime (float timeValue)
```

SetEndTime

Sets the end time value of the event, which is when the event should be processed.

**Syntax**

```
void SetEndTime (float timeValue)
```
GetStartTime

Gets the start time value of this event, which is when the event should be executed.

**Syntax**

```cpp
float GetStartTime () const
```

GetEndTime

Gets the end time value of this event, which is when the event should stop.

**Syntax**

```cpp
float GetEndTime () const
```

GetIsTickEvent

Checks whether this is a tick event.

**Syntax**

```cpp
bool GetIsTickEvent () const
```

ConvertToTickEvent

Converts this event into a tick event.

**Syntax**

```cpp
void ConvertToTickEvent ()
```

GetIsSyncEvent

Checks whether this event is a sync event.

**Syntax**

```cpp
bool GetIsSyncEvent () const
```

SetIsSyncEvent

Specifies whether this event is a sync event.

**Syntax**

```cpp
void setIsSyncEvent (bool newValue)
```

HashForSyncing

Creates a hash on the sync track of a motion.

**Syntax**

```cpp
size_t HashForSyncing (bool isMirror) const
```

For the MotionEvent class source code, see `lumberyard_version\dev\Gems\EMotionFX\Code\EMotionFX\Source\MotionEvent.*`. 
Creating EventData Types

You can create EventData types in a separate gem or directly in a game project's code. Each parameterized motion event can contain more than one EventData object. This makes it possible for users to attach an arbitrary number of EventData objects to a single event.

The EMotionFX::EventData class describes a set of parameters and values that is sent when an event is dispatched. The EventData class is the base class for all event data types. For full source code, see the lumberyard_version\dev\Gems\EMotionFX\Code\EMotionFX\Source\EventData.* files.

To create an EventData type

1. Using the following criteria, subclass either the EventData class or the EventDataSyncable class:
   - For general-purpose parameters, subclass the EventData type.
   - For parameters for the sync track, use the EventDataSyncable class. These parameters are for synchronizing blended motions.

   The following code snippet subclasses the EventDataSyncable class to create a LeftFootEvent for a left footstep.

   ```cpp
   class LeftFootEvent final
   : public EMotionFX::EventDataSyncable
   {
   public:
       AZ_RTTI(LeftFootEvent, "{117454DC-0675-483E-843E-841C57A4354D}",
       EventDataSyncable);
       LeftFootEvent() = default;
       ...
   }
   
   2. Implement the Equal function in the subclass. The Equal function tests whether two EventData instances are equal and is used for deduplication of EventData instances.

   The following example checks whether the EventData passed in is a LeftFootEvent.

   ```cpp
   bool Equal(const EventData& rhs, const bool /*ignoreEmptyFields*/) const override
   {
       // All LeftFootEvents are equal.
       const LeftFootEvent* rhsEvent = azrtti_cast<const LeftFootEvent*>(&rhs);
       return rhsEvent != nullptr;
   }
   
   For more information about the Equal function, see More About the Equal Function (p. 1457).

   3. Implement the Reflect method to reflect the type to the Serialization Context (p. 1033) and Edit Context (p. 1052) contexts.

   When you reflect the event to the edit context, add the Creatable attribute to ClassElement. This makes the EventData type visible in the Animation Editor's Motion Events tab so that users can select it.

   The following code example reflects the LeftFootEvent to the serialize and edit contexts.

   ```cpp
   static void Reflect(AZ::ReflectContext* context)
   {
       AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context);
   }
   ```
if (!serializeContext) return;

serializeContext->Class<LeftFootEvent, EventDataSyncable>()->Version(1);

AZ::EditContext* editContext = serializeContext->GetEditContext();
if (!editContext) return;

editContext->Class<LeftFootEvent>("LeftFootEvent", "")
->ClassElement(AZ::Edit::ClassElements::EditorData, "")
->Attribute(AZ::Edit::Attributes::AutoExpand, true)
->Attribute(AZ::Edit::Attributes::Visibility,
AZ::Edit::PropertyVisibility::ShowChildrenOnly)
->Attribute(AZ_CRC("Creatable", 0x47bff8c4), true);
}
...

4. To add the event to a motion, use the FindOrCreateEventData template, which accepts any subclass of EventData.

... auto footstepData = GetEMotionFX().GetEventManager()->
>FindOrCreateEventData<LeftFootEvent>();
motion->GetEventTable()->FindTrackByName("Sound")->AddEvent(0.2f, footstepData);
...

Example

The following example shows the completed LeftFootEvent sample EventData subclass and the code to add the event to a motion.

class LeftFootEvent final
 : public EMotionFX::EventDataSyncable
{
public:
    AZ_RTTI(LeftFootEvent, "{117454DC-0675-483E-843E-841C57A4354D}", EventDataSyncable);

    LeftFootEvent() = default;

    static void Reflect(AZ::ReflectContext* context)
    {
        AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context);
        if (!serializeContext) return;

        serializeContext->Class<LeftFootEvent, EventDataSyncable>()->Version(1);

        AZ::EditContext* editContext = serializeContext->GetEditContext();
        if (!editContext) return;

        editContext->Class<LeftFootEvent>("LeftFootEvent", "")
            ->ClassElement(AZ::Edit::ClassElements::EditorData, "")
            ->Attribute(AZ::Edit::Attributes::AutoExpand, true)
            ->Attribute(AZ::Edit::Attributes::Visibility,
            AZ::Edit::PropertyVisibility::ShowChildrenOnly)
            ->Attribute(AZ_CRC("Creatable", 0x47bff8c4), true);
    }

    bool Equal(const EventData& rhs, const bool /*ignoreEmptyFields */) const override
    {
        // all LeftFootEvents are equal
        const LeftFootEvent* rhsEvent = azrtti_cast<const LeftFootEvent*>(&rhs);
        return rhsEvent != nullptr;
    }
}
size_t HashForSyncing(bool isMirror) const override { return isMirror ? 1 : 0; }

protected:
    LeftFootEvent(const size_t /*hash*/) : LeftFootEvent() {}

};

auto footstepData = GetEMotionFX().GetEventManager()-
    >FindOrCreateEventData<LeftFootEvent>();

motion->GetEventTable()->FindTrackByName("Sound")->
AddEvent(0.2f, footstepData);

More About the Equal Function

The **Equal** function tests whether two EventData instances are equal.

**Syntax**

```cpp
virtual bool Equal(const EventData& rhs, bool ignoreEmptyFields = false) const = 0;
```

EMotion FX uses the **Equal** method to deduplicate instances of EventData subclasses. The AnimGraphMotionCondition class also uses it for motion event matching logic.

When Event Manager loads a .motion file and deserializes the motion events on the event tracks, EventManager::FindOrCreateEventData processes each EventData instance.

The EventManager stores a list of the EventData instances in use and attempts to find an EventData instance in which the call to Equal(loadedEventData) returns true. If the EventManager finds an EventData instance that is equal, the duplicate data is discarded.

When a call to AnimGraphMotionCondition tests a motion event, AnimGraphMotionCondition::TestCondition calls the **Equal** method with the ignoreEmptyFields parameter set to true. The ignoreEmptyFields parameter enables partial matching of EventData instances. For example, if one of the fields is a string and the string value is empty in the condition, any value in the field matches.

**Synchronizing Blended Motions**

The **EMotionFX::EventDataSyncable** class extends the functionality of the base EventData class and enables events that drive motion synchronization behavior. Use the **EventDataSyncable** class to specify parameters for synchronizing blended motions. The class calls HashForSyncing on the sync tracks of two different motions, compares the results, and finds events that are equal based on their hash value.

For source code, see `lumberyard_version\dev\Gems\EMotionFX\Code\EMotionFX\Source \EventDataSyncable.*`.

**Mirroring**

You can use EMotionFX to mirror motions programmatically. When a motion is being mirrored, its sync events must also be mirrored. To signal this mirroring, the HashForSyncing method accepts an isMirror parameter.

For example, suppose that you use an EventDataSyncable subclass to mirror the gait of a horse. You use an integer field to represent the feet of the horse with the following convention.

```cpp
0=left rear
1=right rear
2=left front
3=right front
```
Implement `HashForSyncing` as in the following example.

```cpp
size_t HashForSyncing(bool isMirror) const
{
    if (!isMirror)
    {
        return m_footIndex;
    }
    // Translate left foot (an even foot index) to right foot, and vice
    // versa
    return (m_footIndex % 2 == 0) ? m_footIndex + 1 : m_footIndex - 1;
}
```

The default implementation for `HashForSyncing` returns the hash of the type ID of the type and ignores the `isMirror` parameter.

## Creating and Simulating a PhysX Ragdoll

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A ragdoll is a physical representation of a character in the animation system that you can use to simulate behavior, such as hit reactions and character death. The physical representation consists of a hierarchy of rigid bodies with simple shapes that are connected by joints. The animation system and the PhysX system work together to simulate the realistic behaviors. While the ragdoll setup occurs in the animation system, the PhysX system is responsible for how a character moves based on environmental interactions and external forces. For example, you can set up your ragdoll so that the character will rotate as it collapses when you apply force to the character's outer shoulder area.

To use the PhysX Ragdoll (p. 757), add it to an entity in Lumberyard Editor. You can then follow the procedures below to create and control the physical representation of the ragdoll.

**Important**

- You can enable the ragdoll to interact physically with the terrain. To do so, add a PhysX Terrain (p. 767) component to an entity in the level.

- The PhysX Character Controller component currently doesn't support the PhysX Ragdoll component.

This topic will teach you how to do the following:

- Set up a ragdoll. (p. 1458)
- Add the ragdoll to an animation graph. (p. 1465)
- Simulate a ragdoll in Lumberyard Editor. (p. 1468)

## Setting Up a Ragdoll

When you set up a ragdoll, you do the following:
• Define joints for the ragdoll. (p. 1459)
• Add ragdoll colliders. (p. 1461)
• Create joint limits. (p. 1463)

In addition, your actor must have a motion extraction node that is a root node. Your ragdoll must have a root node that is a direct parent of the motion extraction node. For example, the Rin character uses root for its motion extraction node. For the ragdoll root node, the Rin character uses 'C_pelvis_JNT, which is a child node of root.

Step 1: Define Joints for the Ragdoll

Do the following to select the joints for your ragdoll.

To select joints for the ragdoll
1. In Lumberyard Editor, choose Tools, Animation Editor.
2. In the Animation Editor, on the right side of the menu bar, choose Physics from the drop-down list. This changes the layout.
3. Choose File, Open Actor and select your actor.
4. In the Skeleton Outliner, multi-select the joints that you want to include in your ragdoll.
5. Right-click one of the selected joints and then choose Ragdoll, Add to ragdoll.

Note
You can add joints to the ragdoll at any time.

6. Click the filter icon next to the search text box, and select Ragdoll joints and colliders. This shows only the joints in the ragdoll and not the full animation skeleton.
The **Skeleton Outliner** displays icons to indicate that:

- A joint is part of the ragdoll
- A joint holds ragdoll colliders
- A joint holds hit detection colliders

7. In the **OpenGL Render window**, use the render options to show or hide the ragdoll colliders (rendered in orange), ragdoll joint limits, and the hit detection colliders (rendered in blue).

   **Note**
   If your ragdoll colliders and hit detection colliders are the same size, you may need to hide the colliders that you are not working on.

8. On the **Ragdoll** tab, you can view and modify the ragdoll properties for the selected joint. For example, the rigid body mass, sleeping threshold, and colliders.
Step 2: Add Ragdoll Colliders

The ragdoll automatically suppresses collisions between joints that are adjacent in the skeleton. This means that adjacent colliders can overlap, but a pair of colliders that are not adjacent should not intersect. If the pair of colliders intersects, they'll collide when the ragdoll is simulated.

In the following example, the collider for the second spine joint (highlighted) can intersect with the first or third spine joint. The first and third spine joints, however, should not intersect.
To add a ragdoll collider

1. In the Animation Editor, on the Ragdoll tab, click Add ragdoll collider and then choose Add box, Add capsule, or Add sphere.
2. On the **Ragdoll** tab, do the following in the collider properties:

   a. Set the **Offset** and **Rotation** to move the collider to the correct location. The **Offset** and **Rotation** are relative to the joint transform.

   b. Adjust the collider dimensions (for example, set the **Height** and **Radius** for a **Capsule**) to resize the collider.

3. Choose **File**, **Save Selected Actors**. This saves the ragdoll data to the `.assetinfo` file for the character. The Asset Processor then bakes the ragdoll data into the `.actor` file.

**Step 3: Create a Joint Limit**

Joint limits are enabled for every joint in your ragdoll. This allows you to edit the joints and ensure they're set up correctly.

The following image shows joint limits that are set up correctly.
Creating and Simulating a PhysX Ragdoll

To create a joint limit

1. In the Animation Editor, in the OpenGL Render Window, adjust the child local rotation so that the red axis points along the bone.
2. Adjust the parent local rotation so that the red axis appears inside the swing cone.
3. Adjust the twist limit value so that the line appears inside the wedge.
Creating and Simulating a PhysX Ragdoll

Adding Your Ragdoll to an Animation Graph

When you create an animation graph to control the ragdoll simulation of your character, you do the following:

- Prepare the actor asset.
- Adjust the animation graph to enable the ragdoll.
- Preview

The animation graph controls the ragdoll simulation of your character. When your character transitions into a blend tree that has a ragdoll node, the ragdoll automatically activates and simulates in game mode in Lumberyard Editor. When your character transitions out of that state, the ragdoll deactivates. The ragdoll node outputs a bind pose in the Animation Editor.

To create an animation graph to transition from running state to ragdoll state

1. In the Animation Editor, on the right side of the menu bar, choose AnimGraph from the drop-down list. This changes the layout.

2. In the Anim Graph pane, click the + icon to create a new animation graph.

3. Right-click the grid and then choose Create Node, Sources, Motion. Alternatively, in the Anim Graph Palette, on the Sources tab, drag Motion into the animation graph.
4. Select the **Motion** node in the animation graph.

5. In the **Attributes** pane, do the following:
   a. For **Name**, enter a name for your motion. For example, **Run**.
   b. Click **Select motions**. In the **Motion Selection** window, select a motion and then click **OK**.

6. Right-click the grid and then choose **Create Node, Sources, Blend Tree**. Alternatively, in the **Anim Graph Palette**, on the **Sources** tab, drag **Blend Tree** into the animation graph.

7. Select the **Blend Tree** node in the animation graph.
8. In the Attributes pane, enter a name for your blend tree. For example, Ragdoll.

9. In the animation graph, connect the Motion node to the Blend Tree node. For example, connect the Run node to the Ragdoll node.

10. Double-click the Blend Tree node.

11. Right-click the grid and then choose Create Node, Sources, Ragdoll. Alternatively, in the Anim Graph Palette, on the Sources tab, drag Ragdoll into the animation graph.

12. Connect the Output Pose for the Ragdoll node to the Input Pose for the Final Node node.

13. At the root of the animation graph, select the transition line that starts from the Motion node and connects to the Blend Tree node. For example, select the transition line that connects the Run node to the Ragdoll node.

   a. In the Attributes pane, click Add condition and then choose Time Condition.

   b. Under Time Condition, set the Countdown Time.
14. In the animation graph, click the **Motion** node for a preview.

## Simulating Your Ragdoll

Once you've created your ragdoll and animation graph, you can simulate the ragdoll in game mode in Lumberyard Editor.

**To simulate your ragdoll**

1. In Lumberyard Editor, right-click the viewport and choose **Create entity**.
2. In the **Entity Inspector**, for **Name**, enter `Ragdoll`.
3. Add an **Actor** component:
   a. Click **Add Component, Actor**.
   b. In the **Actor** component, for **Actor asset**, click the browse (...) button.
   c. In the **Pick EMotion FX Actor** window, select the actor for which you set up the ragdoll and then click **OK**.
4. Add an Anim Graph component:
   
a. Click Add Component, Anim Graph.
   
b. In the Anim Graph component, for Motion set asset, click the browse (...) button.
   
c. In the Pick EMotion FX Motion Set window, select your motion set and then click OK.
   
d. In the Anim Graph component, for Anim graph, click the browse (...) button.
   
e. In the Pick EMotion FX Anim Graph window, select your animation graph and then click OK.
Creating Additive Animations

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Additive animations are animations that you can add as layers on top of a base animation. They are commonly used to add a partial-body animation to a full-body animation. Because additive animations do not interfere with base animation functionality, you can reuse the same base to create many variations in motion. Reusing the base also has the advantage of greatly reducing the overall asset count.

Because additive animations preserve the underlying animation's style, they are useful for adding poses and animations to a character's upper body. For example, you can use additive animations to make a character breathe, look around, flinch, or change posture. This adds variety to the animations and avoids what might otherwise be a monotonous look.

You can create additive animations in two ways:

- At runtime, use the Blend Two Additive and Pose Subtract nodes. For more information, see Blend Two Additive Node (p. 1425) and Pose Subtract Node (p. 1427).
- At asset processing time, use the Asset Browser and FBX settings motion additive modifier to convert an existing animation into an additive animation. This approach removes the generation of additive animations from runtime processing and improves runtime performance. See the following procedure.

To convert a motion into an additive animation

1. From Lumberyard Editor, choose Tools, Asset Browser.
2. In the Asset Browser, locate the FBX animation file that you want to convert into an additive animation.
3. Right-click the FBX animation file and choose Edit Settings.
4. In the FBX Settings dialog box, on the Motions tab, click Add Modifier, and then choose Additive Motion.
5. In the Motion additive section, for Base frame, specify the number of the frame that contains the reference pose that you want to subtract.

   For example, if you're creating an additive animation for the idle state, the base frame can be the first frame of your idle motion. If you're creating an additive aim blendspace, the base frame can be the frame that contains the center aim.

   **Note**
   The reference pose that you want to subtract must be in the FBX file that you selected.

---

### Retargeting Motions

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Animation Editor to retarget motions from one actor to another actor. This lets you reuse motions that you've created for an actor, and to quickly prototype motions on other actors while you wait for new ones to be created. The retarget feature is also important to ensure that actors are scaled appropriately. Without this feature, the retargeted actor stretches and scales to the size of the original actor for which the motion was recorded. With this feature, the retargeted actor retains its size.

For example, you may have a human character that's six feet tall and your motions are recorded for that actor. You may also have a giant character that's 18 feet tall. If you play the motions for the human character on the giant character without retargeting, the giant character will scale to the size of the human character. When you enable the retarget feature, the giant character retains its height of 18 feet.

**Important**
To retarget motions, your assets **must** meet these requirements:

1. The first frame in every motion must be the actor's bind pose. For example, if you have an animation that contains frames 1-100, the actor's bind pose (skinned mesh and bones) must be added at frame 0.
2. The bone names and hierarchy **must** be identical between the original actor and the retargeted actor. The retargeted actor **may** contain additional leaf bones in its hierarchy, but the areas of the hierarchy that are being retargeted must be identical.

   **Note**
   If your retargeted actor has the same bone names and transforms as the original actor, you do not need to use the retarget motions procedure.

**To retarget motions**

1. In your DCC tool, add the actor's bind pose, which includes skinning and bones, to the first frame of your motion. Then export the file into Lumberyard. For more information, see Importing Assets into Lumberyard (p. 263).
2. Open Lumberyard Editor.
3. In the Asset Browser, right-click the file that you imported, and choose Edit Settings.
4. In the Fbx Settings window, on the Motions tab, choose Add Modifier, Motion range.
   Note
   If you do not see the Motion range modifier, be sure that your .fbx file has keyframes.
5. Under Motion range, do the following:
   a. Set the Start frame to the frame that does not include the bind pose. For example, if the original actor's bind pose is on frame 0, set the Start frame to 1. This ensures that the bind pose does not play in your motion.
   b. Set the End frame to the last frame of your motion. Based on the example in the previous step, the last frame would be 100.
   c. Click Update.
      Note
      If you receive an error, check the number of keyframes in your motion, and update the End frame.

The frame settings are shown in the following example:

6. Repeat steps 1 to 5 for all of the motions that you want to retarget.
7. In Lumberyard Editor, choose Tools, Animation Editor.
8. In the Animation Editor, in the center pane, on the Anim Graph tab, open your animation graph (p. 1349).
9. In the right pane, on the Attributes tab, select the Retarget check box.
Creating Simulated Objects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you animate a character (actor), your actor might wear objects that move differently from the primary motion. For example, if your actor runs and wears a backpack, the backpack might sway back and forth. To create this dynamic movement, you create a simulated object. A simulated object acts as a container for the bones on your actor. In the Animation Editor, you can specify how loose objects, such as chains, backpacks, or long hair, move in relation to the actor.

**Note**
Simulated objects don't collide with ragdoll colliders or PhysX entities in the level.

In this procedure, you'll do the following:

1. Create a simulated object that's attached to an actor.
2. Add colliders to the skeleton so that the object collides with the actor's body.
3. Adjust your settings so that the animation appears more smooth and realistic.
4. View the simulation in the render window.
Prerequisites

Before you begin, you must do the following:

- Using Project Configurator, enable the Samples Project (p. 146).
- In the Animation Editor, complete the following:
  - Import your actor, such as the rinActor.fbx file
  - Create a motion set
  - Import your motions, such as the rin_Run.fbx

For more information, see Step 1: Creating a Motion Set (p. 1354).

Topics

- Setting Up a Simulated Object (p. 1474)
- Setting Up the Simulated Object Anim Graph Node (p. 1476)
- Setting Up Simulated Object Colliders (p. 1480)
- Using Debug Mode to Refine the Simulation (p. 1483)
- Using Parameters to Adjust the Animation During Runtime (p. 1487)

Setting Up a Simulated Object

In the following procedure, add motion to a tassel attached to the character.

To set up a simulated object

1. In the Animation Editor, choose Layouts and choose SimulatedObjects.
2. In the Skeleton Outliner, select the bones to add to a simulated object. Enter a name in the search bar or filter for results.
3. Select the bones, right-click, and choose Simulated Object, Add selected joints, <New simulated object>.

Example

The L_tassle_01_JNT bone and its children are selected.

4. In the dialog box, enter a name for the simulated object and click OK.
Example

The simulated object appears in the **Simulated Object** panel.

5. In the **Simulated Objects** panel, select an object or joint. The properties that you can customize appear in the **Simulated Object Inspector** panel.

   **Note**
   After you create an anim graph, you can preview the simulation. For now, keep the default values.

Example

The first joint in the simulated object chain (L_tassle_01_JNT) is **Pinned**. Pinned joints follow the original joint. If you have multiple joints, you can pin them if you don't want them to move as a simulated object. The root joint is always pinned.

6. In the render window, view the collider radius of each joint in the simulated object. Select the simulated object to view all colliders or select a joint to see an individual collider.

   **Tip**
   - In the render window, click the icon to toggle the collider.
   - The **Collision radius** is the distance in which the simulated object joint avoids a collider specified in the simulated object. If the value is 0, the joint remains on the surface of the collider.
   - After you complete the next section, you can adjust the **Collider radius** to get your preferred results.
Creating Simulated Objects

Example

7. To save your actor, click the save icon in the Actor Manager.

Example

Setting Up the Simulated Object Anim Graph Node

After you set up the simulated object, create an anim graph to view the simulation. This lets you preview how the tassel moves in relation to the actor's primary motion.

To set up an anim graph node for the simulated object

1. In the Anim Graph panel, click the + icon to create an anim graph.
2. Right-click the graph and choose Create Node, Sources, Blend Tree.
Example

3. Double-click the BlendTree0 node, right-click the anim graph grid, and choose Create Node, Physics, Simulated Object.

Example

4. Connect the Pose output of the SimulatedObject0 node to the Input Pose of the FinalNode0 node.
5. Right-click the anim graph and choose Create Node, Sources, Motion.
6. Connect the Output Pose of the Motion0 node to the Pose input of the Simulated Object0 node.
Example

Your graph should look like the following.

7. Select the **Motion0** node and in the **Attributes** panel, click the + icon to add a motion to the node.
8. In the **Motion Selection Window**, select a motion, such as the *run*, and click **OK**.
9. Save the anim graph and enter a name, such as *simulatedobjects*.animgraph.
10. Select the **SimulatedObject0** node and in the **Attributes** panel, click **Select simulated objects**.

Example

11. In the dialog box, select the simulated object.
12. Save the anim graph.

**Example**

In the Anim Graph panel, click the play icon to run the anim graph. The colliders that you set up for each joint don't collide with other colliders and the tassel goes through the arm and chest of the actor.

In the next procedure, you create another collider attached to the actor's skeleton. This prevents the tassel from moving into the actor's body.
Creating Simulated Objects

Setting Up Simulated Object Colliders

In the following procedure, add a collider to the spine and left arm to reduce the tassel’s movement. This ensures that the tassel has something to collide with, which prevents it from moving through the actor’s body.

To set up simulated object colliders

1. In the Skeleton Outliner, select the joint or joints to add a simulated object collider to. In this example, use the C_spine_04_JNT joint.
2. Right-click your selection and choose Simulated object collider, Add collider, Add capsule. This creates a shape that defines the collider area for the joint.

Example

You can also add a sphere if that shape fits your actor better.
In the **Skeleton Outliner**, a simulated object collider icon appears next to the joint.

3. In the **Simulated Object Inspector**, adjust the capsule so that it's larger than the actor's geometry.

   In this example, change the **Height** to 0.344 and the **Radius** to 0.162.

**Example**

By default, the name of the joint (C_spine_04_JNT) is also the name of the collider.

4. Repeat **Step 1** to add the L_arm_JNT joint. This creates another simulated object collider to the upper-left arm.

5. Repeat **Step 2** and **3** to set up a collider for the arm.

6. Adjust the capsule so that it fits the arm. In this example, enter the following values:

   - **Height** to 0.322
   - **Radius** to 0.081
   - **Rotation**: X to 180
   - **Radius**: Y to 89.99
• **Radius:** Z to 180

**Example**

By default, the name of the joint (`L_arm_JNT`) is also the name of the collider.

![Simulated Object Inspector](image)

7. In the **Skeleton Outliner**, select the joints where the simulated object colliders are attached.

**Tip**

In the render window, deselect the first icon (**Solid**) and select the second (**Wireframe**) to view the capsule colliders.

**Example**

In the render window, the colliders appear purple. If you deselect the joint, the collider appears gray.

Now that you added the simulated objects to the spine and arm, add these colliders to the simulated object.
8. In the **Simulated Objects** panel, select the simulated object.

9. In the **Simulated Object Inspector**, for **Collides with**, select the available colliders. This enables the simulated object joints (the tassel) to collide with the actor's body (the spine and left arm).

**Example**

![Simulated Object Inspector](image)

10. In the **Actor Manager**, save the actor. You might have to wait for Asset Processor to finish processing your changes.

**Using Debug Mode to Refine the Simulation**

In the following procedure, use the debug mode to refine the simulated object's movement.

**To use debug mode for the simulation**

1. In the **Anim Graph** panel, click the play icon to run the anim graph.
Example

Use the speed slider to reduce the speed of the animation. Note that the tassel moves through the neck and arm of the actor.

2. In the SimulatedObject0 node, click the right-corner box. This enables debug rendering for the node.

Example

3. In debug mode, you can do the following.
   a. To turn off the simulated object colliders, click the icon.
b. To turn off joint collision radius, click the icon.

Example

In debug mode, the following appear in the render window:

- Simulated object colliders – The colliders that the simulated object collides with. The colliders appear in light gray.
- Joint angle limit cones – The angle of a joint that simulates. The cones appear in light pink.
- Joint collision radius – The size of the collision radius on a joint. The radius appears in dark gray.

In the render window, you can click the first icon to toggle the actor geometry and see only simulated objects.

4. To fix the simulated object (tassel) movement, do the following.
   a. In the Simulated Object panel, select L_tassle_02_JNT.
   b. In the Simulated Object Inspector, select Pinned. This pins the second part of the tassel and prevents it from moving.
   c. For Joint angle limit, enter 80 to increase the swing of the joint.
   d. In the Simulated Object panel, select L_tassle_03_JNT.
   e. In the Simulated Object Inspector, for Radius, enter 0.065.
5. To fix the simulated joint colliders (chest and arm of the actor), do the following.
a. In the **Skeleton Outliner** panel, select the `C_spine_04_JNT` where you added the simulated joint collider.

b. In the **Simulated Object Inspector**, for **Radius**, enter `0.0.172`.

c. In the **Skeleton Outliner** panel, select the `L_arm_JNT` where you added the simulated joint collider.

d. In the **Simulated Object Inspector**, for **Radius**, enter `0.0.101`.

6. If you're satisfied with the results, save the actor.

7. To disable debugging mode, click the upper-right box again on the `SimulatedObject0` node.

**Example**

The following animation is the finished debugged version of the actor running. Note that the tassel no longer passes through the actor's body.
Using Parameters to Adjust the Animation During Runtime

You can adjust the SimulatedObject node to change its animation during runtime. To do so, create a Parameter node and attach them to the node in the anim graph. If you don't, the SimulatedObject node uses the properties that you entered in the Attributes panel.

To adjust the animation during runtime

1. In the Parameters panel, click the + icon and choose Add parameter.
2. Enter the following values.
   a. For Value type, select the Float (slider).
   b. For Name, enter Stiffness.
   c. For Default, enter 1.0.
   d. For Minimum to 1.0.
   e. For Maximum, enter 100.

Example

Your parameter should look like the following.
3. Click Create.

4. Repeat Steps 2 and 3 but change Name to Gravity, Default to 1.0, Minimum to 0 and Maximum to 5.

5. Repeat Steps 2 and 3 but change Name to Damping, Default to 1.0, Minimum to 1 and Maximum to 100.

6. Repeat Steps 2 and 3 but change Name to Weight, Default to 1, Minimum to 0 and Maximum to 1.

**Example**

Your parameters should look like the following.
7. In the Anim Graph grid, right-click and choose Create Node, Sources, Parameters.
8. On the Parameters node, connect the Stiffness output to the Stiffness factor, the Gravity output to the Gravity factor, and so on.

Example

Your graph should look like the following.

9. Pay the anim graph and adjust the sliders for the Parameters to view your changes.

Note
The simulated object, simulated joints, and parameters on the anim graph share the following properties: Stiffness, Gravity, Damping
When you adjust the properties, the Animation Editor calculates the results of these properties using the following:

- Stiffness factor parameter \( x \) simulated object stiffness \( x \) simulated joint stiffness
- Gravity factor parameter \( x \) simulated object gravity \( x \) simulated joint gravity
- Damping factor parameter \( x \) simulated object damping \( x \) simulated joint damping

Add Cloth Colliders to actors

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Applying cloth simulation to actors adds realism and dynamism in their every movement. Adding cloth colliders greatly enhances the effect. Cloth colliders can prevent cloth from penetrating into the meshes of the actor. To add cloth colliders, you must enable the NVIDIA Cloth Gem.

For more information, see the NVIDIA Cloth Gem documentation (p. 1182).

**Add Cloth Colliders to an actor**

1. In Lumberyard Editor, choose **Tools, Animation Editor**.
2. In the **Animation Editor**, on the right side of the menu bar, choose **Physics** from the drop-down list. This changes the layout.
3. Choose **File, Open Actor**, and select your actor.
4. In the **Skeleton Outliner**, select the joint that you want to add the cloth collider to.
5. Right-click the selected joint, and then choose **Cloth, Add collider**. You can add colliders (capsules or spheres). You can also copy them from Hit Detection, Ragdolls or Simulated Objects.
6. The Cloth Colliders tab shows you the list of cloth colliders of the selected joint.

7. Adjust the dimensions, offset, and rotation of cloth colliders.
8. Choose **File, Save Selected Actors**.

### Best Practices for Working with Skinned Meshes for Animation

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

See the following guidelines when working with skinned meshes for the **Animation Editor**.

**Topics**

- Creating Skinned Meshes for Actors (p. 1492)
- Exporting Actors and Motions (p. 1494)
- Tutorials: Importing FBX Files as Actors and Motions (p. 1497)
- Troubleshooting Skinned Meshes (p. 1497)
- Using Actor LODs to Optimize Game Performance (p. 1498)

### Creating Skinned Meshes for Actors

Use the following best practices when you create your character for the **Animation Editor**. In Lumberyard, a character is a skinned mesh.

#### Setting up the World Coordinate System and Root Joint

If you use the y-up or z-up world coordinate system in your DCC, use the following guidelines to set up your character:
Use a root joint for your skinned mesh. This is required to ensure that motion extraction works properly in the **Animation Editor**.

Do not use transforms, groups, or parent nodes in the hierarchy above your root joint. The root joint must be the top parent of the skeletal hierarchy to ensure that motion extraction works properly.

Set the root joint position at the origin: 0,0,0.

Set the root joint rotation and orientation to 0,0,0.

Orient your character so that the front orthographic camera view shows the front of your character.

When using the Lumberyard **Animation Editor**, ensure that your imported character faces the positive y direction. The Asset Processor automatically adds a *Coordinate system change* modifier in the .fbx settings. The default value for **Facing Direction** is **Rotate 180 degrees around the up axis**. This enables the game entity's forward direction and character to point in the same direction.

If your character faces the negative y direction after you import the character into the **Animation Editor**, in the **Asset Browser** right-click your .fbx file and choose **Edit Settings**. For the **Coordinate system change** modifier, set **Facing Direction** to **Do Nothing**.

Use the same *Coordinate system change* modifier for the actor and each of the actor's motions. This is required to ensure that animations work properly in the **Animation Editor**.

### Setting up Skin Binding

Observe the following best practices when you set up skin binding:

- Delete the geometry history on your mesh before skinning the geometry to joints.
- When skinning your mesh, limit your maximum influences per vertex to four. Lumberyard currently supports only four weight influences per vertex using the .fbx pipeline.
- Skin bind your mesh at the origin and in the same forward direction as the root joint.
- Check the bind pose before exporting your skinned mesh. For example, if the mesh moves after unbinding, you must reskin the mesh in order to prevent any errors in Lumberyard.
- Ensure that your skinned mesh has one bind pose in your DCC before you export to an .fbx file.
- Do not include any static meshes with your skinned meshes. Lumberyard cannot render unskinned meshes that are parented to bones.

Do the following to reskin your mesh:

**To reskin your mesh in Maya**

1. For **Menu Set**, choose **Rigging**.
2. Click **Skin, Unbind Skin**.
3. In the **Detach Skin Options** dialog box, for **History**, choose **Delete history**. Click **Detach** and then click **Close**.
4. Move or rotate your mesh to the appropriate position.
5. Click **Modify, Freeze Transformations**.
6. In the **Freeze Transformations Options** dialog box, select the **Translate, Rotate**, and **Scale** check boxes. Click **Freeze Transform** and then click **Close**.
7. Select your mesh.
8. Choose **Edit, Delete by Type, History**.
9. In the outliner, select the bones that you want skinned. Hold **Ctrl** and select the mesh that you want skinned.
10. Click **Skin, Bind Skin**.
11. In the **Bind Skin Options** dialog box, for **Bind to** choose **Selected joints**. For **Bind method**, choose your preferred bind method. For **Skinning method**, choose **Dual quaternion** or **Linear**. Click **Bind Skin** and then click **Close**.
Note
You must export and save your deformer weights in order to import the weight maps after reskinning.

To reskin your mesh in 3ds Max
1. Select your mesh.
2. On the Modify tab, expand Advanced Parameters. Click Save.
3. Save the mesh envelope (skin).
4. On the Modify tab, select the skin. Right-click the skin and choose Delete.
5. On the Utilities tab, click Reset XForm.
6. Click Reset Selected.
7. Click Collapse.
8. Click Collapse Selected.
10. For Modifier List, choose Skin.
11. Under Advanced Parameters, click Load and choose the mesh envelope (skin) that you saved earlier.
   Note
   You must save your weights in order to load the weights after adding a new skin modifier.
12. In the Load Envelopes dialog box, click Match by Name to preserve the weights that you saved earlier. Close the dialog box.

If you want to add a root joint to a skinned mesh, follow the preceding steps to unbind and reskin your mesh.

Exporting Actors and Motions
Use the following best practices when you export your skinned meshes using the FBX Settings tool.

- If you use the z-up world coordinate system, use the following guidelines:
  - Ensure that your DCC scene is set to z for the up-axis world coordinate system.
  - Set the axis conversion up axis to z when exporting an .fbx file.
- If you use the y-up world coordinate system, use the following guidelines:
  - Ensure that your DCC scene is set to y for the up-axis world coordinate system.
  - Set the axis conversion up axis to y when exporting an .fbx file.
- When you create an .actor file, export your skinned mesh at the bind pose without any keyframes.
- When you create a .motion file, bake animations before you export. Alternatively, bake animations when you use your DCC's .fbx export tools.
- Export only the skeleton and mesh. Do not use transforms, groups, or parent nodes in the hierarchy above your root joint. The root joint must be the top parent of the skeletal hierarchy to ensure that motion extraction works properly.
- Remove unused geometry, bones, vertices, materials, and nodes that are not necessary for the .fbx asset. This reduces the processing time and offers a better chance that the automatic processing works properly without making adjustments later. In your DCC, consider naming nodes with _ignore as a suffix to prevent those nodes from being processed.
- We recommend that you enable the EMotion FX Animation (p. 1124) gem or the CryLegacyAnimation gem, but not both. You can disable the animation gem that you don't use. For more information, see Add modular features and assets with Gems (p. 1064).
Use the following guidelines for vertex count:

**If you enabled the CryLegacyAnimation gem:**

Asset Processor processes your .fbx file to create an .actor file. Occasionally, the number of vertices does not match the original vertex count of the .fbx file. Lumberyard allows a maximum of 65,536 vertices for skinned geometry after asset processing. However, the original vertex count from your DCC tool cannot be up to 65,536 vertices. As a best practice when modeling your geometry, divide 65,536 by 3 for a value of 21,845 vertices. This allows you to import .fbx files that have 21,845 or fewer vertices. After you export the .fbx file, verify that Asset Processor successfully processes the skinned mesh. If Asset Processor does not successfully process the skinned mesh, reduce the number of vertices in the model until the file is successfully processed.

**If you enabled the EMotion FX Animation (p. 1124) gem:**

The theoretical maximum number of vertices for skinned meshes after processing is 4,294,967,295 \((2^{32} – 1)\). Although this limit is exceptionally high, we recommend that you follow best practices when modeling your skinned meshes. Use a reasonable polygon count that's suitable for the game device for which you are developing. Experiment with the number of polygons to get the desired quality while balancing game performance. You may need to adjust the polygon count if your game displays many actors at once. The value range for polygon counts can vary between 1,000 to 30,000. The polygon count that you should use depends on the game device, and the performance and quality of the actor.

The more vertices in an .fbx file, the longer Asset Processor takes to process it. View Asset Processor often and check for errors.

### Using the Maya Game Exporter

The following are typical settings for the Maya Game Exporter when you export your skinned mesh character into Lumberyard.

**Export settings for .actor files**

- Use the **Model** tab to export your .fbx files.

  ![Game Exporter](image)

  - Use the following settings:
    - Select **Export Selection** from the drop-down list.
• Select the **Skinning** check box.
• If you are exporting blendshapes (morph targets), select the **Blendshapes** check box.
• Clear the **Animation** check box. If this is selected, a .motion file is created.
• Select your world coordinate system from the **Up Axis** drop-down list.
• Navigate to a save path and specify the name of your .fbx file.
• When you're done choosing these settings, select all of the bones for the character's skeleton and all of the skinned meshes.
• Click **Export**.

**Export settings for .motion files (Animation Clips tab)**

• Use the **Animation Clips** tab to export your .fbx files.

- Use the following settings:
  - Select **Export Selection** from the drop-down list.
  - Click the + button to add an animation clip.
  - For **Clip Name**, enter a name for the clip.
  - For **Start** and **End**, specify a frame.
  - Select the **Bake Animation** check box.
  - Select your world coordinate system from the **Up Axis** drop-down list.
  - Navigate to a save path and specify the name of your .fbx file.
  - When you're done choosing these settings, select all of the bones for the character's skeleton.
  - If you are exporting blendshapes with your animation, also select the skinned mesh that has the blendshape animations. For example, if your character's face mesh has blendshapes, select the skeleton and the character's face mesh.
• Click **Export**.

**Export Settings for .motion files (Time Editor tab)**

• Use the **Time Editor** tab to export your .fbx files.
Best Practices for Working with Skinned Meshes for Animation

Use the following settings:

- Select **Export Selection** from the drop-down list.
- Select the clip that you want to export from the **Time Editor Clips** drop-down list.
- Click the + button to add an animation clip.
- Select the **Bake Animation** check box.
- Select your world coordinate system from the **Up Axis** drop-down list.
- Navigate to a save path and specify the name of your .fbx file.
- When you're done choosing these settings, select all of the bones for the character's skeleton.
- If you are exporting blendshapes with your animation, also select the skinned mesh that has the blendshape animations. For example, if your character's face mesh has blendshapes, select the skeleton and the character's face mesh.
- Click **Export**.

**Tutorials: Importing FBX Files as Actors and Motions**

To learn about importing FBX files as actors into Lumberyard, watch the following video tutorial.

**Importing Characters into Lumberyard as Actors**

To learn about importing FBX files as motions into Lumberyard, watch the following video tutorial.

**Importing Animations into Lumberyard as Motions**

For more information, see Customize FBX asset export with FBX Settings (p. 409).

**Troubleshooting Skinned Meshes**

If your character's skinning appears visually broken in the **Animation Editor**, the following issues could be why:

- The **Coordinate system change** modifier isn't identical for the .actor and .motion files
- You might need to reset the bind pose
To change the coordinate system modifier

1. In Lumberyard Editor, in the Asset Browser, navigate to your .fbx file.
2. Right-click the .fbx file and choose Edit Settings.
3. In the FBX Settings window, for the Coordinate system change modifier, set Facing Direction to Do Nothing or Rotate 180 degrees around the up axis.
4. Ensure your .actor and .motion files use the same Coordinate system change modifier.

To reset a bind pose in Maya

1. Ensure your character is in the bind pose.
2. In Maya, select the character root bone.
3. In the Rigging menu, click Skin and then select the Go to Bind Pose check box.
4. For the Input box, choose Select by name.
5. Search for all bind poses by entering bindPose* in the search box.
6. Press Enter to select all of the bind poses and then press Delete.
7. Select the root bone of your character.
8. In Maya, in the bottom left corner, enter the following to reset the bind pose for your character:
   `dagPose -bp -save;`

Using Actor LODs to Optimize Game Performance

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use levels of detail (LODs) to optimize the rendering performance of games. This is especially true for large in-game worlds, which are constrained by hardware, frame rate, and the real-time rendering limits. LODs increase performance and reduce draw calls by displaying progressively less detail in objects that are farther from the camera. To further reduce draw calls, lower LODs have multiple textures combined into one texture.

Topics

- Using Actor LODs in Lumberyard (p. 1498)
- Using Digital Content Creation Tools to Create LODs (p. 1499)
- Using the FBX Settings Tool to Check LODs (p. 1501)
- Adding Actor LOD to a Level in Lumberyard (p. 1503)
- Adding the Simple LOD Distance Component (p. 1504)

Using Actor LODs in Lumberyard

In Lumberyard, you can use up to five actor LODs in addition to your actor’s base mesh. An LOD of 1 has the highest level of detail, and an LOD of 5 the least. Each successive LOD typically has its vertices reduced by 50 percent from the previous level and reduces the number of materials used.

Features

In Lumberyard, you can use actor LODs on skinned meshes, skeletons, and materials or textures. Lumberyard performs the following actions:

- Provides a Simple LOD Distance component that supports rendering of different LODs.
• Reads and imports skinned mesh LODs from a single FBX file.
• Auto populates skinned mesh LODs that use a soft naming convention.

Requirements and Limitations

Note the following requirements:

Skinned Meshes
• Each LOD mesh must skin to the same skeleton. However, you can choose which bones to skin to.

Skeletons
• To skin each LOD mesh to a different skeleton, create a separate skeletal hierarchy.
• For each LOD, the skeleton hierarchy must remain the same. Remove leaf bones first and work up the chain.
• In lower LODs, you cannot remove in-between bones. For instance, if you have Spine1, Spine2 and Spine3 for an LOD of 1, you cannot delete Spine2 and leave Spine1 and Spine3.

Materials or Textures
• Each actor requires a separate material group.
• Create a material LOD in the same way that you create a material for static meshes.

Using Digital Content Creation Tools to Create LODs

To author actor LODs in a DCC tool like Autodesk Maya or Autodesk 3ds Max, you can use one of the following two methods:

• Soft naming conventions
• LOD groups

**Important**
Currently, Lumberyard supports skeletons created with LOD groups only.

To use the soft naming convention to create LODs

1. Create an actor with multiple LOD meshes by adding a suffix at the end of the name of each LOD mesh. See the following table for examples.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lod1</td>
<td>Rin_lod1</td>
</tr>
<tr>
<td>_lod2</td>
<td>Rin_lod2</td>
</tr>
<tr>
<td>_lod3</td>
<td>Rin_lod3</td>
</tr>
<tr>
<td>_lod4</td>
<td>Rin_lod4</td>
</tr>
<tr>
<td>_lod5</td>
<td>Rin_lod5</td>
</tr>
</tbody>
</table>

For more information, see FBX soft naming conventions (p. 443).
2. Follow the instructions in Customize FBX asset export with FBX Settings (p. 409) to export the meshes to FBX format so that you can use them in Lumberyard.

**To use LOD groups to create actor LODs**

- To create an actor with multiple LOD meshes, use an empty LOD group as the parent of the skeleton and the LOD meshes. The following example shows a skeleton created with an LOD group and imported into Lumberyard.
Using the FBX Settings Tool to Check LODs

After you export your .fbx file to Lumberyard, do the following:
To use the FBX Settings to check LODs

1. In Lumberyard Editor, choose **Tools, Asset Browser**.
2. Right click the .fbx file and choose **Edit Settings**. The **FBX Settings** tool shows a **Level of Detail** modifier that lists the additional LOD meshes.
You can use the **FBX Settings** tool to verify that your LODs were imported correctly. Currently, you can also unselect a mesh or skeleton for each LOD, but you can't move a mesh or skeleton across a level or add a new mesh or skeleton to a level.

**Adding Actor LOD to a Level in Lumberyard**

After you import your LODs into Lumberyard Editor, you can add an actor LOD to a level.

**To add an actor LOD to a level in Lumberyard**

1. In Lumberyard Editor, choose **Tools, Asset Browser**.
2. Select and drag the exported `.fbx` file to the viewport. In the **Actor** component that appears in the **Entity Inspector**, a shared material file with multiple materials contains the LOD asset.
3. To preview the LODs, change the number for **LOD Level** in the **Actor** component.

**Example**

![Actor LOD Preview](image)

**Notes**

- Lumberyard supports only one material group for each actor.
- Author the material definition (.mtl) file with all the actor LOD submaterials inside the same .mtl file.
- If you author in Maya, you can assign a different shader for each LOD mesh or use the same shader for all the meshes.
Adding the Simple LOD Distance Component

To enable rendering of levels of detail in Lumberyard, you must add the Simple LOD Distance component to your level. The Simple LOD Distance component performs a simple LOD distance check. The LOD distance is the distance that the current level of detail must reach before it changes to the next LOD. Each distance must be greater than the previous distance.

To add the Simple LOD Distance component to your level

1. In Lumberyard Editor, choose Tools, Entity Inspector.
2. In Entity Inspector, click Add Component.
3. In the Animation section, select the Simple LOD Distance component.
4. Specify an LOD distance or use the default.
5. In the viewport, zoom out. Notice that the actor mesh changes for each level of detail when the specified LOD distance is reached.

Adding particle effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes an advanced particle effects system that you can use to simulate explosions, fire, smoke, sparks, water spray, fog, snow, rain, and other effects. Use the Particle Editor to create and manage particle effects in your game. You can place particle emitters in your level, link them to an object, set up a material to define a custom effect, and control these effects from Script Canvas and the Track View.

Lumberyard uses two shaders for rendering particles:

- Particles Shader (p. 1704) – Render particle effects that are affected by light. These effects can cast shadows and cause reflections.
- ParticleImposter Shader (p. 1704) – Render mesh particle effects that are not affected by light. These effects do not cast shadows or cause reflections.

Topics

- Using the Particle Editor (p. 1504)
- Particle Effects Best Practices (p. 1576)
- Advanced Particle Techniques (p. 1576)
- Particle Debugging with Console Variables (p. 1578)

Using the Particle Editor

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
Use the **Particle Editor** to create, edit, preview, and save particle libraries and emitters that are used with entities in your levels.

**To open the Particle Editor**

- In Lumberyard Editor, choose **Tools, Particle Editor**. You can also click the particle editor icon in the Lumberyard Editor toolbar.

The **Particle Editor** includes the following UI elements:

- **Libraries** panel – Lists the particle art assets. You can view and interact with multiple libraries simultaneously.

- **Preview** panel – Displays the selected, active particle effect. The preview camera automatically positions to capture the entire particle. Use the W, A, S, and D keys and mouse controls to pan, zoom, and rotate the camera.

- **Attributes** panel – Lists the properties for the selected particle.
Using the Particle Editor

• **Level of Detail** panel – Displays level of detail (LOD) information for added particles. For more information, see Managing Particle Level of Detail (LOD) (p. 1570).

**Adding a Particle Component**

**To add a particle component to your level**

• Do one of the following:
  • Drag a particle emitter from the Particle Editor library and drop it into the Lumberyard Editor viewport.
  • Drag a particle effect asset from the Asset Browser into your scene. Choose the emitter that you want to use from the Entity Inspector properties. For more information, see Entity Inspector (p. 475).
  • Create a Particle component entity and assign the particle effect with the Particle effect library parameter. For more information, see the Particle (p. 702) component and Adding Components to an Entity (p. 479).

**Customizing the Particle Editor UI**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Particle Editor has a customizable interface so you can set up your workspace in different layouts. All panels are moveable. You can undock panels to become floating windows, or you can dock panels in new locations to rearrange the look and feel. Save your custom layouts to access later or to share with others.

You can customize the Particle Editor in the following ways:

- **Floating panels**
  Drag the panel title bar to separate the panel from the editor and make it a floating window.

- **Docking panels**
  Dock panels on the edges of the window or anywhere an orange highlight appears. To dock a floating panel, drag the panel title bar into the Particle Editor window.

- **Tabbing panels**
  You can dock a panel inside another panel to create a tabbed view. Toggle the tabs to display one panel and hide the other. This option minimizes the number of panels that are visible at a time.

- **Showing or hiding panels**
  Customize which panels are visible by clicking View and choosing a panel to show or hide.

- **Resetting the layout**
  To reset the Particle Editor layout, click View, Reset to Default Layout. This positions the Libraries panel on the top left, the Preview panel on the middle left, the Level of detail panel on the bottom left, and the Attributes panel on the right.

- **Exporting a layout**
  You can export and share custom layouts. After you create a layout, click View, Export layout. Navigate to the location where you want to export the layout file.
Importing a layout

To import custom layouts, click View, Import layout. Locate and select the layout file.

Managing Particle Libraries

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Particle libraries are .xml assets that contain the data for all effects in the library. The .xml files are located in the \game_project_name\libs\particles directory.

Use the Particle Editor to add and manage particle libraries. You can rename library assets by duplicating and renaming a library in the Particle Editor. Do not rename the base .xml file; doing so can negatively affect the entities in your levels.

Multiple selection

Press Ctrl or Shift to select multiple emitters. The emitters can be from different libraries. You can do any of the following:

- Copy – Copies the selected item. If you copy multiple items, they become children of the item that you paste them on.
- Delete – Deletes all selected emitters.
- Group – Groups the selected items that share the same parent. Select the emitters that you want to group and press Ctrl+G. Alternatively, you can right-click the selected emitters and choose Group.

If you use hot keys while multiple items are selected, the hot keys apply to all selected items.

Drag and drop

Add emitters from any of the libraries into a specific library. To do this, drag and drop the emitters onto the library name. You can also drag emitters from the library to a new parent in the same library.

Search

Type queries into the Libraries panel search box to instantly view results. Click the arrow icon on the left to display previous search results.

Level library (Deprecated)

The level library is no longer updated. You can import particle data stored in level files that were created in Lumberyard 1.8 and earlier. To do so, open a level and, in the Particle Editor, choose File, Import Level Library. This imports the data to a project particle library in the \game_project_name\libs\particles directory.

We recommended that you update all entities that reference the level library emitters to use this newly created library.

Topics

- Adding Particle Libraries (p. 1508)
- Importing Particle Libraries (p. 1508)
- Saving Particle Libraries (p. 1508)
- Using Particle Libraries (p. 1509)
Adding Particle Libraries

Particle effect data is stored in an XML-based library file.

To add a particle library

1. In the Particle Editor, do one of the following:
   - Choose Create new library.
   - Choose File, Add library.
   - Choose the drop-down arrow in the Libraries panel, and choose Add library.
   - Choose the icon to the right of the search bar.

2. In Libraries, where the default name is highlighted, enter a name for the library.
3. Choose Add Particle or Add Folder and then enter a name for the new particle or folder.

Importing Particle Libraries

Particle effect data is stored in an XML-based library file. You can import a particle library for your game.

To import a particle library

1. In the Particle Editor, choose File, Import.
2. In the Pick Particles dialog box, select a library to load. Choose OK. You can also choose the drop-down arrow to access the menu.

Saving Particle Libraries

You can save your particle library in the Particle Editor.

To save a particle library

- In the Particle Editor, right-click the library name and choose Save. You can also press Ctrl+S to save all libraries that are loaded in the editor.
Your library is now saved to the `\game_project_name\libs\particles` directory. You can nest libraries in any hierarchy in the `\libs\particles` directory.

**Using Particle Libraries**

The **Particle Editor** provides a context menu with the following options for you to manage your particle libraries. Right-click a library name to access the context menu.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add New</td>
<td><strong>Add Particle</strong> – Adds a new emitter to the library. The default keyboard shortcut is <strong>Ctrl+N</strong>.</td>
</tr>
<tr>
<td></td>
<td><strong>Add Folder</strong> – Adds a new folder to the library. The default keyboard shortcut is <strong>Ctrl+Alt+N</strong>.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves changes to the selected library. Libraries are saved in the <code>\game_project_name\libs\particles</code> directory.</td>
</tr>
<tr>
<td>Duplicate</td>
<td>Duplicates and renames the library.</td>
</tr>
<tr>
<td>Disable/Enable All</td>
<td>Disables or enables all items in the library.</td>
</tr>
<tr>
<td>Expand/Collapse All</td>
<td>Expands or collapses all branches in the library. You can click a library name to collapse or expand the entire library. The contents do not lose their collapsed or expanded state when you do so.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the library from the library list. The library list is still available on disk.</td>
</tr>
<tr>
<td>Reload</td>
<td>Reloads the library.</td>
</tr>
</tbody>
</table>
You can access the following toolbar menu items and buttons in the **Particle Editor** and **Libraries** panel.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add Particle</strong></td>
<td>Adds a particle effect. By default, the particle effect is a child of the selected particle, folder, or library. Set the particle name in the <strong>New Particle Name</strong> window.</td>
</tr>
<tr>
<td><strong>Add Folder</strong></td>
<td>Adds a directory in the library so you can organize your particle effects.</td>
</tr>
<tr>
<td><strong>Add library</strong></td>
<td>Adds a particle library.</td>
</tr>
<tr>
<td><strong>Import</strong></td>
<td>Opens the file browser to import the selected particle libraries.</td>
</tr>
<tr>
<td><strong>Import Level Library</strong></td>
<td>Imports particle data stored in level files that were created in Lumberyard 1.8 or earlier.</td>
</tr>
<tr>
<td><strong>Save All</strong></td>
<td>Saves all modified particle libraries to disk.</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>Closes the <strong>Particle Editor</strong>.</td>
</tr>
<tr>
<td><strong>Undo</strong></td>
<td>Clears the last change.</td>
</tr>
<tr>
<td><strong>Redo</strong></td>
<td>Removes the last undo.</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>Copies all of the settings for the selected item to the clipboard.</td>
</tr>
<tr>
<td><strong>Paste</strong></td>
<td>Writes data from the clipboard to the selected item.</td>
</tr>
<tr>
<td><strong>Duplicate</strong></td>
<td>Duplicates the selected particle effect.</td>
</tr>
<tr>
<td><strong>Add LOD</strong></td>
<td>Adds a level of detail (LOD) to the selected emitter.</td>
</tr>
<tr>
<td><strong>Rename</strong></td>
<td>Renames the selected item.</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>Deletes the selected item.</td>
</tr>
<tr>
<td><strong>Reset to default</strong></td>
<td>Resets all properties and parameters for the selected item to the default values and states.</td>
</tr>
<tr>
<td><strong>Edit Hotkeys</strong></td>
<td>Opens the <strong>HotKey Configuration</strong> window for you to edit keyboard shortcuts (hotkeys).</td>
</tr>
</tbody>
</table>

### Using the Preview Panel

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Preview** panel in the **Particle Editor** to view the particle effects.
This panel has the following features:

A. Display content in the viewport
B. Display the main menu
C. Choose which emitter hierarchies to display in the viewport
D. Reset the viewport to default settings
E. Toggle time of day visibility to simulate the approximate time of day
F. Toggle the display of the particles wireframe
G. Play back the timeline
H. Play, pause, and step forward
I. Reset emitter playback
J. Loop playback

You can access the following Preview panel options in the main menu and context menu.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import Mesh</td>
<td>Imports a game mesh to view.</td>
</tr>
<tr>
<td>Play</td>
<td>Toggles playback.</td>
</tr>
<tr>
<td>Pause</td>
<td>Pauses playback.</td>
</tr>
<tr>
<td>Step forward</td>
<td>Steps forward one frame.</td>
</tr>
<tr>
<td>Play loop</td>
<td>Toggles the playback from once to looping.</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets the playback to the beginning.</td>
</tr>
<tr>
<td>Reset camera position</td>
<td>Resets the preview camera to the default position.</td>
</tr>
<tr>
<td>Show/Hide overdraw</td>
<td>Toggles display of overdraw in the editor.</td>
</tr>
<tr>
<td>Wireframe view</td>
<td>Toggles display of the particles wireframe.</td>
</tr>
<tr>
<td>Show/Hide particle count</td>
<td>Toggles display of the particle count.</td>
</tr>
<tr>
<td>Show/Hide bounding box</td>
<td>Toggles display of the emitter bounding box.</td>
</tr>
<tr>
<td>Show/Hide gizmo</td>
<td>Toggles display of the preview window transform gizmo.</td>
</tr>
<tr>
<td>Show/Hide playback controls</td>
<td>Toggles display of the preview playback controls.</td>
</tr>
<tr>
<td>Show/Hide grid</td>
<td>Toggles display of the preview grid.</td>
</tr>
<tr>
<td>Show/Hide emitter shape</td>
<td>Toggles display for the visualization of shape emitters and their relative coordinates.</td>
</tr>
<tr>
<td>Background color</td>
<td>Opens the color picker for you to set the color of the preview background.</td>
</tr>
<tr>
<td>Grid color</td>
<td>Opens the color picker for you to set the color of the preview grid.</td>
</tr>
<tr>
<td>Reset colors</td>
<td>Resets the background and grid to the default colors.</td>
</tr>
<tr>
<td>Move on spline</td>
<td>Displays a spline in the Preview panel for the emitter to move along. This helps to preview the emitter motion.</td>
</tr>
</tbody>
</table>

Spline mode
- Single path – Traverses the path once
Using the Particle Editor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Loop</td>
<td>Traverses the path indefinitely</td>
</tr>
<tr>
<td>• Ping-pong</td>
<td>Traverses back and forth on the path</td>
</tr>
<tr>
<td><strong>Spline shape</strong></td>
<td></td>
</tr>
<tr>
<td>• Line</td>
<td></td>
</tr>
<tr>
<td>• Sine wave</td>
<td></td>
</tr>
<tr>
<td>• Coil</td>
<td></td>
</tr>
<tr>
<td><strong>Spline speed</strong></td>
<td></td>
</tr>
<tr>
<td>• Speed x1</td>
<td></td>
</tr>
<tr>
<td>• Speed x2</td>
<td></td>
</tr>
<tr>
<td>• Speed x3</td>
<td></td>
</tr>
<tr>
<td>• Speed x5</td>
<td></td>
</tr>
<tr>
<td><strong>Reset spline settings</strong></td>
<td>Disables all spline settings.</td>
</tr>
<tr>
<td><strong>Reset to default</strong></td>
<td>Resets the preview window to the default configuration.</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>Closes the Preview panel.</td>
</tr>
</tbody>
</table>

**Particles Attributes Reference**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In the **Particle Editor**, you can modify how emitters and particles look and behave. You can find all available parameters in the Attributes pane. Except for a parameter's base value, most numeric parameters allow random variation over a particle or emitter lifetime.

Some parameters have additional settings:

- **Random** – Specifies how much a particle's parameter value deviates from the default value of 0 (no variation).
- **Strength Over Emitter Life** – Controls the alpha strength over the lifetime of the particle. This parameter works with finite particles only. This parameter has no effect if set to continuous.
- **Strength Over Particle Life** – Controls the alpha strength over the lifetime of an individual particle. For example, you can use this parameter to make a smoke particle fade to nothing once its lifetime has finished. The particle fades out earlier or later depending on where you reduce the value to zero.

**Topics**

- Using the Curve Editor in the Particle Editor (p. 1514)
Using the Particle Editor

- Advanced Attribute (p. 1514)
- Audio Attribute (p. 1516)
- Collision Attribute (p. 1517)
- Comment Attribute (p. 1521)
- Configuration Attribute (p. 1521)
- Emitter Attribute (p. 1522)
- Lighting Attribute (p. 1528)
- Movement Attribute (p. 1529)
- Particles Attribute (p. 1532)
- Rotation Attribute (p. 1540)
- Size Attribute (p. 1541)
- Visibility Attribute (p. 1542)

Using the Curve Editor in the Particle Editor

Use the Curve Editor to edit the attribute values over the emitter and particle lifetimes. Strength Over Emitter Life is active only if you have an emitter lifetime that is greater than zero.

To edit the curves

1. In the Curve Editor, double-click along the curve timeline to add a key.
2. Click and drag the keyframe to the preferred value and shape.

Advanced Attribute

In the Advanced attribute, specify advanced appearance and optimization settings.
### Advanced Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Force Generation</td>
<td>(CPU only) Adds a force that is generated by the emitter.</td>
</tr>
<tr>
<td></td>
<td>- <strong>None</strong> – Does not add a force.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Wind</strong> – Creates a physical wind force, approximately following the velocity, direction, volume, and timing of the emitter's particles. This wind affects all particles and objects in its region, except particles in the emitter group. Setting the emitter's <strong>Speed</strong> to a negative value creates a wind force in the opposite direction. You can use this to create a vacuum force.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Gravity</strong> – Creates a physical gravity force similar to wind, but creates a gravitational acceleration force instead of wind velocity.</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>None</strong></td>
</tr>
<tr>
<td>Fill Rate Cost</td>
<td>(CPU only) Multiplies the emitter's contribution to the total fill rate. This affects automatic culling of large particles when the global limit is reached. Set this to a value greater than 1 if the effect is relatively expensive or unimportant. Set this to a value less than or equal to 0 if the effect is important. You should not experience automatic culling.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Heat Scale</td>
<td>(CPU only) Multiplies thermal vision.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 4</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Sort Quality</td>
<td>(CPU only) Specifies more accurate sorting of new particles in the emitter's list. To avoid the popping that results from changing the render particle order, particles are never re-sorted after emission. Instead they are sorted only when emitted, based on the main camera's position.</td>
</tr>
</tbody>
</table>
Using the Particle Editor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>0 (default, fastest) – Places the particle at the front or back of the list, depending on its position relative to the center of the emitter bounding box. Additional force is not added.</td>
<td></td>
</tr>
<tr>
<td>1 (medium slow) – Sorts existing particles into a temporary list. New particles do a quick binary search to find an approximate position.</td>
<td></td>
</tr>
<tr>
<td>2 (slow) – Sorts existing particles into a temporary list. New particles do a full linear search to find the position of least sort error.</td>
<td></td>
</tr>
</tbody>
</table>

Valid values: 0 to 2
Default value: 0

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half Res</td>
<td>Renders particles in a separate, half resolution pass, reducing rendering cost. To enable this parameter, set the <code>r_ParticlesHalfRes</code> console variable to 1. For more information, see Using the Console Window (p. 210).</td>
</tr>
<tr>
<td>Default value: false</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streamable</td>
<td>(CPU only) Allows texture or geometry assets to stream from storage as normal.</td>
</tr>
<tr>
<td>Default value: true</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Size Discard</td>
<td>(GPU only) Discards particles below the defined screen space pixel size. Valid values: 0 to 255</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth of Field Blur</td>
<td>(GPU only) Blurs particles against depth of field full screen effects. This excludes geometry and decal types. Default value: false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spherical Volume</td>
<td>(CPU only) Simulates adding spherical volume to the particle and multiplies particle size to define the volume. Default value: 1</td>
</tr>
</tbody>
</table>

**Audio Attribute**

In the **Audio** attribute, specify which sounds the particle system emits and when.
Audio Attribute Parameters for CPU Emitters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Trigger</td>
<td>Selects the start trigger sound asset to play with the emitter.</td>
</tr>
<tr>
<td>Stop Trigger</td>
<td>Selects the stop trigger sound asset to play with the emitter.</td>
</tr>
<tr>
<td>Sound FXParam</td>
<td>Applies the modulate value to the sound. The effect depends on how you define the particlefx parameter for the individual sound. Depending on the sound, this value might affect volume, pitch, or other attributes. You can set a Random value and Strength Over Emitter Lifetime curve. Valid values: 0+ Default value: 1</td>
</tr>
</tbody>
</table>
| Sound Control Time    | • **EmitterLifeTime** – Plays for the length of the emitter's lifetime.  
• **EmitterExtendedLifeTime** – Plays for the length of the emitter's lifetime plus all particles' lifetimes (until all particles die).  
• **EmitterPulsePeriod** – Plays for the length of the pulse period. |

Collision Attribute

In the **Collision** attribute, specify how to control the particle's physical setup.
## Using the Particle Editor

### CPU Collision Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physics Type</strong></td>
<td>Specifies how the particle interacts physically.</td>
</tr>
<tr>
<td></td>
<td>- <strong>None</strong> – No collisions or other physics. This setting is used by default.</td>
</tr>
<tr>
<td></td>
<td>- <strong>SimpleCollision</strong> – Collides the particle with the static environment using simple physics. This is the simplest mode.</td>
</tr>
<tr>
<td></td>
<td>- <strong>SimplePhysics (Geometry particles)</strong> – Creates a particle in the physics system and collides using a spherical particle model. Use <strong>Surface Type</strong> to set the bounciness.</td>
</tr>
<tr>
<td></td>
<td>- <strong>RigidBody (Geometry particles)</strong> – Creates a particle in the physics system and collides using the full collision mesh of the assigned geometry. This is the most expensive mode. Use <strong>Surface Type</strong> to set the bounciness.</td>
</tr>
<tr>
<td><strong>Collide Terrain</strong></td>
<td>Includes terrain in particle collisions.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td><strong>Collide Static Objects</strong></td>
<td>Includes non-terrain, static objects in particle collisions. This is expensive.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td><strong>Collide Dynamic Objects</strong></td>
<td>Includes non-terrain, dynamic objects in particle collisions. This is expensive.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
</tbody>
</table>
### Using the Particle Editor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>On Collide</strong></td>
<td>Upon impact with the static environment, the particle dies.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Die</strong> – The particle dies upon colliding with an object.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Bounce</strong> – If you enable <strong>Collide Terrain</strong>, <strong>Collide Static Objects</strong>, or <strong>Collide Dynamic Objects</strong>, the particle bounces off the respective type. If none of these are enabled, the selected particle ignores the object that it collides with and passes through. This is useful with <strong>Spawn Indirection – Parent Collide</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Stop</strong> – The particle stops when it collides with an object.</td>
</tr>
<tr>
<td><strong>Max Collision Events</strong></td>
<td>Limits the number of collisions that the particle can have in its physics simulation. This affects only particles that have <strong>Physics Type</strong> set to <strong>Rigid Body</strong>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 255</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Bounciness</strong></td>
<td>Controls the elasticity for collision response. This affects only particles that have <strong>Physics Type</strong> set to <strong>Simple Collision</strong>. You can override this feature by setting <strong>Surface Type</strong>. You can also use a special value of −1 to have the particle die on first collision.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Collision Fraction</strong></td>
<td>Determines the fraction of emitted particles that perform collisions.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td><strong>Collision Cutoff Distance</strong></td>
<td>Specifies the maximum distance from the camera at which collisions are performed. A value of 0 means infinite.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Surface Type</strong></td>
<td>Selects from a variety of surface material types for the collision behavior. If set, <strong>Surface Type</strong> overrides <strong>Bounciness</strong> and <strong>Dynamic Friction</strong>.</td>
</tr>
<tr>
<td></td>
<td>Default value: none</td>
</tr>
<tr>
<td><strong>Dynamic Friction</strong></td>
<td>The coefficient of dynamic friction. If set, <strong>Surface Type</strong> overrides <strong>Dynamic Friction</strong>. This affects only particles that have <strong>Physics Type</strong> set to <strong>Simple Collision</strong>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
</tbody>
</table>
### Using the Particle Editor

**Name**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls the fraction of the particle's visible radius to use for the physical radius. This affects particles that have <strong>Physics Type</strong> set to <strong>Simple Physics</strong> and geometry particles that have <strong>Physics Type</strong> set to <strong>Simple Collision</strong>.</td>
</tr>
<tr>
<td>Valid values: 0+</td>
</tr>
<tr>
<td>Default value: 1</td>
</tr>
</tbody>
</table>

**Name**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls the particle density in kg/m³. An example of a physically correct value is Water = 1000. This affects only particles that have <strong>Physics Type</strong> set to <strong>Simple Physics</strong> or <strong>Rigid Body</strong>.</td>
</tr>
<tr>
<td>Valid values: 0+</td>
</tr>
<tr>
<td>Default value: 1000</td>
</tr>
</tbody>
</table>

### GPU Collision Attribute Parameters

**Name**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies how the particle interacts physically.</td>
</tr>
<tr>
<td><strong>None</strong> – No collisions or other physics. This setting is used by default.</td>
</tr>
<tr>
<td><strong>Frame Buffer</strong> – Collides with the screen space depth buffer.</td>
</tr>
<tr>
<td><strong>Cubemap</strong> – Collides with a depth cubemap buffer that is generated around the emitter.</td>
</tr>
</tbody>
</table>

**Name**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets the far plane distance for cubemap depth buffer generation.</td>
</tr>
<tr>
<td>Default value: 20</td>
</tr>
</tbody>
</table>

**Name**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upon impact with the static environment, the particle dies.</td>
</tr>
<tr>
<td><strong>Die</strong> – The particle dies upon colliding with an object.</td>
</tr>
<tr>
<td><strong>Bounce</strong> – The particle bounces off the object that it collides with.</td>
</tr>
<tr>
<td><strong>Stop</strong> – The particle stops when it collides with an object.</td>
</tr>
</tbody>
</table>
Using the Particle Editor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bounciness</strong></td>
<td>Controls the elasticity for collision response. This affects only particles that have <strong>Physics Type</strong> set to <strong>Simple Collision</strong>. You can override this feature by setting <strong>Surface Type</strong>. You can also use a special value of -1 to have the particle die on first collision.</td>
</tr>
<tr>
<td><strong>Thickness</strong></td>
<td>Controls the fraction of the particle's visible radius to use for the physical radius. This affects particles that have <strong>Physics Type</strong> set to <strong>Simple Physics</strong>.</td>
</tr>
</tbody>
</table>

Valid values: any
Default value: 0

**Comment Attribute**

Save or edit comments about an emitter in the **Comment** text box.

**Configuration Attribute**

In the **Configuration** attribute, specify the minimum and maximum specs for your particle emitters.
Configuration Attribute Parameters for CPU Emitters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Config Min</td>
<td>Specifies the minimum system configuration level for the effect. If the configuration is lower than the set value, the item will not appear. Choose from Low, Medium, High, or VeryHigh. Default value: Low</td>
</tr>
<tr>
<td>Config Max</td>
<td>Specifies the maximum system configuration level for the effect. If the configuration is higher than the set value, the item will not appear. Choose from Low, Medium, High, or VeryHigh. Default value: VeryHigh</td>
</tr>
<tr>
<td>Platforms</td>
<td>Specifies which platform to use with the effect.</td>
</tr>
<tr>
<td></td>
<td>• PCDX11</td>
</tr>
<tr>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• macOS GL</td>
</tr>
<tr>
<td></td>
<td>• macOS Metal</td>
</tr>
<tr>
<td></td>
<td>Default value: all selected (true)</td>
</tr>
</tbody>
</table>

**Emitter Attribute**

In the Emitter attribute, specify how to control the location and spawning attributes of the particle and emitter shape. The parameters are updated based on the type that you select.
Emitter Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Type</td>
<td>Specifies whether the emitter is a CPU or GPU type.</td>
</tr>
<tr>
<td>Emitter Shape Type</td>
<td>Specifies the emitter shape type.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Angle</strong> – Standard emitter type based on angle parameters. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Point</strong> – Spherical distribution emitter.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sphere</strong> – Sphere shape emitter with particles controlled along the sphere's circumference coordinates.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Circle</strong> – Circle shape emitter with particles controlled along the circumference and vertical coordinates.</td>
</tr>
</tbody>
</table>
### Name
<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Box</strong> – Box shape emitter with particles controlled along the planes of the box.</td>
</tr>
<tr>
<td><strong>Trail</strong> – Emitter to create trail type effects.</td>
</tr>
<tr>
<td><strong>Beam</strong> – Emitter to create beam type effects.</td>
</tr>
</tbody>
</table>

**Note**
If you change the **Emitter Shape Type**, the attributes will reset to the default settings for the selected emitter. In the confirmation message, you can select **Do not prompt again**. This prompt is reset when you select **Reset to default layout**.

### Relative Particle Movement

Determines the particle motion in the emitter's space. The following is an example of particles that are emitted upward from an emitter:

- **No** – The emitted particles stay in world space and fall behind as the emitter moves away, for example, like smoke from the chimney of a train.
- **Yes** – The emitted particles stay relative to the emitter's local space and won't fall behind as the emitter is moved. An example is a smoke column going straight up from the chimney and staying vertical with the emitter. This excludes tail particles.
- **Yes with Tail** – The emitted particles stay relative to the emitter's local space, including any associated tail particles.

Default value: **No**
### Parameter Inheritance

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Inheritance</td>
<td>Specifies the source for default (starting) effect parameters.</td>
</tr>
<tr>
<td>• <strong>Standard</strong> (default) – Uses the values set in DefaultParticleEmitters.xml.</td>
<td></td>
</tr>
<tr>
<td>• <strong>System</strong> (deprecated) – Reads the System.Default.xml file to use as defaults for the angle emitter. If this effect does not exist, you can use standard defaults. You can generate the System.Default.xml file by adding a library named System and an emitter named default. Save the library to your <code>&lt;project&gt;/libs/particles</code> directory.</td>
<td></td>
</tr>
<tr>
<td>• <strong>Parent</strong> – Uses the parent particle effect for defaults. This is useful when creating a parent effect with one set of parameters and a variety of subeffects that alter some of the parameters for variation. Subeffects can spawn on their own. Edit the parent effect to update the default values of all subeffects.</td>
<td></td>
</tr>
</tbody>
</table>

The selected source has the following consequences:

• When you create a new effect, default parameters are set from the following file:

  \`lumberyard_version\dev\Editor\Plugins\ParticleEditorPlugin\defaults\DefaultParticleEmitters.xml\`.

• Changing the inheritance source does not change any other parameters. However, different parameters may be highlighted if their defaults have changed.

To reset parameters to the default values that are located in the DefaultParticleEmitters.xml file, right-click the parameter and choose **Reset to default**. You can modify the default values for the project using the same file.

When you save effects to .xml libraries, only non-default values are saved. When you load effects from the .xml libraries, the current default values for the effect's inheritance are used as a base.

When you edit a parent effect's parameters, the non-edited parameters of all children (and descendants) that have **Parent** selected are instantly updated.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spawn Indirection</strong></td>
<td>This parameter has the following values:</td>
</tr>
<tr>
<td></td>
<td>• Direct – Spawns without relying on the parent's input for timing. This is the default value.</td>
</tr>
<tr>
<td></td>
<td>• ParentStart – Spawns once the parent has spawned.</td>
</tr>
<tr>
<td></td>
<td>• ParentCollide – Spawns a particle with this setting when the parent particle has collided with an object.</td>
</tr>
<tr>
<td></td>
<td>• ParentDeath – Spawns a particle with this setting when the parent particle has lived out its lifetime.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This parameter is supported only within the same CPU or GPU particle type.</td>
</tr>
<tr>
<td><strong>Attach Type</strong></td>
<td>Specifies the emission location when the parent emitter has geometry.</td>
</tr>
<tr>
<td></td>
<td>• None – Particles ignore geometry and emit from the emitter center as normal.</td>
</tr>
<tr>
<td></td>
<td>• Bounding Box – Particles emit from the object's bounding box.</td>
</tr>
<tr>
<td></td>
<td>• Physics – Particles emit from the geometry of the attached physics object (mesh or simple primitive).</td>
</tr>
<tr>
<td></td>
<td>• Render – Particles emit from the full mesh of the render object (static or animated mesh). This is generally more CPU-intensive than emitting from physics.</td>
</tr>
<tr>
<td>Default value:</td>
<td><strong>None</strong></td>
</tr>
<tr>
<td><strong>Attach Form</strong></td>
<td>Specifies the elements of the geometry from which particles emit.</td>
</tr>
<tr>
<td></td>
<td>• None – Not active.</td>
</tr>
<tr>
<td></td>
<td>• Vertices – Emits randomly from the vertices of the geometry. This is the most efficient form of mesh emission.</td>
</tr>
<tr>
<td></td>
<td>• Edges – Emits randomly from the edges of the geometry. This is useful for effects on breaking element pieces.</td>
</tr>
<tr>
<td></td>
<td>• Surface – Emits randomly from the surfaces (faces) of the geometry.</td>
</tr>
<tr>
<td></td>
<td>• Volume – Emits randomly inside the volume of the geometry.</td>
</tr>
<tr>
<td>Default value:</td>
<td><strong>Vertices</strong></td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>Specifies the total number of particles that are active at a given time and determines the emission rate (\textit{Count/Particle Lifetime}). You can set a Random value and the Strength Over Emitter Life curve.</td>
</tr>
<tr>
<td>Valid values:</td>
<td>(0^+)</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **Maintain Density**| Increases the emission rate (and particle count) when an emitter moves in order to maintain the same spatial density as when motionless. You can scale the increase from 0 to 1.  
  • **Reduce Alpha** – When **Maintain Density** is active, this reduces particle alpha in order to maintain the same overall emitter alpha.  
  Valid values: 0+                                                                 |
| **Continuous**      | If false, emits all particles at once and then dies. If true, emits particles gradually over the emitter lifetime. If true and **Emitter Lifetime** = 0, emits particles gradually at a rate of **Count/Particle Lifetime** per second, indefinitely.  
  Default value: false                                                                 |
| **Spawn Delay**     | Delays the start of the emitter for the specified time. This is useful to delay subeffects relative to the overall emitter creation time. Can set a Random value.  
  Valid values: 0+  
  Default value: 0                                                                 |
| **Emitter Lifetime**| If **Continuous** = true, specifies the lifetime of the emitter. Emitter lifetime does not apply to noncontinuous effects, which always disappear as soon as they have emitted all of their particles. You can set a Random value.  
  Valid values: 0+  
  Default value: 0 (infinite lifetime)                                                                 |
| **Pulse Period**    | If greater than 0 and **Continuous** = false, restarts the emitter repeatedly at this interval. You can set a Random value.  
  Valid values: any  
  Default value: 0                                                                 |
| **Orient to Velocity** | Forces the particle x-axis to align to the velocity direction. You can use Rotation parameters to rotate the particle further.  
  Default value: false                                                                 |
| **Position Offset** | XYZ values define the spawning position away from the emitter itself in emitter space.  
  Valid values: any  
  Default values: 0, 0, 0                                                                 |
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Random Offset   | XYZ values define the range of a random spawning box in both directions, away from the position offset.  
                  | Valid values: any                                                           
                  | Default values: 0, 0, 0                                                     |
| Offset Roundness | Specifies the fraction of spawning volume corners to round.                  
                  | Valid values: 0 (box shape) to 1 (ellipsoid shape)                          
                  | Default value: 0                                                           |
| Offset Inner Fraction | Specifies the ratio of inner to outer spawning volume.                    
                  | Valid values: 0 (spawn within entire volume) to 1 (spawn only at surface)   
                  | Default value: 0                                                           |

**Lighting Attribute**

In the **Lighting** attribute, specify how to control the particle lighting.

![Lighting Attribute](image)

**Lighting Attribute Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Light Source| (CPU only) Causes each particle to create a deferred light, where color is equal to the **Color** value.  
                  | • **Affects This Area Only** – Use with clip volumes. When enabled, the particle lights do not exceed the volume boundary.  
                  | • **Radius** – Radius of the light. You can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves.  
                  | • **Intensity** – Intensity of the light. You can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves. |
### Using the Particle Editor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffuse Lighting</td>
<td>Multiplies the particle color for dynamic (diffuse) lighting.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: false, 0, 0</td>
</tr>
<tr>
<td>Diffuse Backlighting</td>
<td>Specifies the fraction of diffuse lighting that is applied to unlit particle directions.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 (standard diffuse and normals facing the light are lit the most) to 1 (omnidirectional diffuse and light affects all normals equally).</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Emissive Lighting</td>
<td>Multiplies the particle color for constantly emitting light. You can add a value to make a particle appear as if it's glowing.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Environment Probe Lighting</td>
<td>(CPU only) Controls the amount of diffuse lighting that is contributed from environment probes.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Receive Shadows</td>
<td>(CPU only) Allows shadows to cast on the particles.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>Cast Shadows</td>
<td>(GPU and geometry particles only) Allows particles to cast shadows.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>Not Affected by Fog</td>
<td>(CPU only) Causes particles to ignore scene fog.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
</tbody>
</table>

**Note**
When there is a single light source, some or all particles can appear unlit (black). To address this:

1. Add an environment probe to the scene to create indirect lighting. For more information, see the [Environment Probe (p. 587)](environmentprobe) component.
2. Specify a value for the **Environment Probe Lighting** parameter to enable environment probe lighting on the particle system. For example, a value of 0.5 applies light from the environment probe at half intensity.

### Movement Attribute

In the **Movement** attribute, specify how to control the particle's movement.
For the Air Resistance, Gravity Scale, Turbulence 3D Speed, Turbulence Size, and Turbulence Speed parameters, you can set a Random value and Strength Over Emitter Lifetime and Strength Over Particle Lifetime curves.

Movement Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min visible segment length</td>
<td>Trail Emitter (CPU) (p. 1560) only) Tail particles are visible only when they have moved the specified distance.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>Min visible distance</td>
<td>Trail Emitter (CPU) (p. 1560) only) Specifies the minimum distance between the start and end of a trail segment. Segments that are smaller than this value become transparent.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Speed</td>
<td>Specifies the initial speed of particles. You can set a Random value and Strength Over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>Acceleration</td>
<td>XYZ values define the constant acceleration that is applied to particles in world space.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Default value: 0, 0, 0</td>
<td></td>
</tr>
<tr>
<td>Inherit Velocity</td>
<td>Specifies the fraction of initial velocity that is inherited from the particle's parent. For indirect particles, the parent particle's velocity is inherited. For direct particles, the emitter's velocity is inherited.</td>
</tr>
<tr>
<td>Valid values: any</td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Bind Emitter to Camera</td>
<td>(CPU only) Forces the emitter to relocate to the main camera's position. This is useful (with Space Loop) for making a rain or snow effect, which the player cannot pass.</td>
</tr>
<tr>
<td>Default value: false</td>
<td></td>
</tr>
<tr>
<td>Space Loop</td>
<td>(CPU only) Loops particles within a region around the camera, as defined by Camera Min/Max Distance (under the Visibility tab). This is useful for making a rain or snow effect, which has an effective infinite spawning area.</td>
</tr>
<tr>
<td>Default value: false</td>
<td></td>
</tr>
<tr>
<td>Air Resistance</td>
<td>Particles behave as if encountering resistance and slow down over time.</td>
</tr>
<tr>
<td>Valid values: 0+</td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Gravity Scale</td>
<td>Applies multiple of world gravity to particles. Set most physicalized particles to 1 (use air resistance to provide drag). Set the parameter to a negative value for buoyant particles such as smoke.</td>
</tr>
<tr>
<td>Valid values: any</td>
<td>Default value: 0, 0, 0</td>
</tr>
<tr>
<td>Turbulence 3D Speed</td>
<td>Adds a 3D, random, turbulent movement to the particle, with the specified average speed.</td>
</tr>
<tr>
<td>Valid values: 0+</td>
<td>Default value: 0                                                                ----------------------------------------------------------------</td>
</tr>
<tr>
<td>Turbulence Size</td>
<td>Adds a spiral movement to the particle, with the specified radius. Set the axis of the spiral from the particle's velocity.</td>
</tr>
<tr>
<td>Valid values: 0+</td>
<td></td>
</tr>
<tr>
<td>Turbulence Speed</td>
<td>When Turbulence Size is greater than 0, specifies the angular speed, in degrees/second, of the spiral motion.</td>
</tr>
<tr>
<td>Valid values: any</td>
<td>Default value: 0                                                                ----------------------------------------------------------------</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
**Target Attraction** | Specifies how particles behave if the emitter is attached to a target. By default, all particles are attracted to any target to which the emitter is linked. These parameters customize that behavior.

- **Target**
  - **External** – Particles are attracted to a target entity, if the emitter is linked to one (default).
  - **OwnEmitter** – Particles are attracted to their emitter’s origin.
  - **Ignore** – Particles ignore any external attractor.
- **Extend Speed** – Particles speed up to reach the target in their lifetime. Otherwise, they move at a real-world, natural speed toward the target and may not reach it.
- **Shrink** – Particles shrink as they approach the target.
- **Orbit** – Particles orbit around the target when reached. Otherwise, they disappear into the target.
- **Radius** – Distance from the target that particles orbit around or disappear. You can set a Random value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves.

Valid values: any
Default value: 0

**Particles Attribute**

In the **Particles** attribute, specify how to control the particle’s basic appearance.

**Note**
We recommend that you set up the **Particles** attribute first because it includes the **Texture** file, which is used for most particles.
## Particles Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Particle Life Time</strong></td>
<td>Specifies the lifetime of individual particles. After an emitter's lifetime has expired, spawned particles live out their own lifetime. Valid values: 0+</td>
</tr>
<tr>
<td><strong>Remain While Visible</strong></td>
<td>Indicates that particles do not die until the entire emitter is out of view. This is typically used for emitters that move through space quickly and die to avoid particle pooling and other unintended consequences. Default value: false</td>
</tr>
<tr>
<td><strong>Facing</strong></td>
<td>Determines how the sprite or geometry is oriented in space. You can further modify texture orientation using the following rotational parameters.</td>
</tr>
</tbody>
</table>
## Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera (default)</td>
<td>Faces the viewer, with texture X and Y aligned with screen X and Y. In this mode only, particles are assumed to represent spherical objects and are lit accordingly. In all other modes, particles are lit as flat polygons.</td>
</tr>
<tr>
<td>CameraX</td>
<td>Rotates about the local y-axis only and faces the camera as much as possible.</td>
</tr>
<tr>
<td>Free</td>
<td>Rotates freely in 3D. Be sure to give it some rotation. The default orientation is equal to the emitter's orientation.</td>
</tr>
<tr>
<td>Horizontal</td>
<td>Faces horizontal to the emitter's z-axis.</td>
</tr>
<tr>
<td>Velocity</td>
<td>Faces the direction of movement.</td>
</tr>
<tr>
<td>Water</td>
<td>Faces upward and is moved and aligned to the nearest water plane.</td>
</tr>
<tr>
<td>Terrain</td>
<td>Faces upward and is moved and aligned to the nearest terrain location.</td>
</tr>
<tr>
<td>Decal</td>
<td>Renders the particle as a deferred decal, projected onto the nearest surface along the entity's y-axis. The Thickness parameter under Collision controls the projection depth. This parameter works only with materials and does not work with textures.</td>
</tr>
<tr>
<td>Shape</td>
<td>Faces the nearest edge of the emitter shape boundary. You can use this parameter with the following shape types: Circle, Sphere, and Box Emitter.</td>
</tr>
</tbody>
</table>

Default value: Camera

## Blend Type

Applies to 2D particles only and determines how the sprite blends with the background.

- **Alpha Based** – Final color = Particle color * Particle alpha + Background color * (1 – Particle alpha)
- **Additive** – Final color = Particle color + Background color
- **Multiplicative** – Final color = Particle color * 2 * Background color
- **Opaque** – Final color = Particle color

Default value: Alpha Based

## Material

Opens the Asset Browser for you to assign a material for the 2D sprite particles. Different shaders are uniquely affected by the lighting and environment.

Default value: Empty

## Sorting Method

(GPU only) Enables intra-GPU emitter sorting.

- **None** – No sorting
- **Bitonic** – Sorts using a bitonic sorting algorithm
- **OddEven** – Sorts using an odd even merge sorting

Default value: None
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting Convergence</td>
<td>(GPU only) Sorts convergence per frame for an odd to even merge sort.</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Texture</td>
<td>Opens the Asset Browser to specify a texture for the 2D sprite particles.</td>
</tr>
<tr>
<td></td>
<td>When you pause on the input box, a preview of the texture appears.</td>
</tr>
<tr>
<td></td>
<td>Default value: /Editor/Plugins/ParticleEditorPlugin/defaultparticle.dds</td>
</tr>
<tr>
<td>Normal Map</td>
<td>(GPU only) Opens the Asset Browser to specify a normal map for the 2D GPU</td>
</tr>
<tr>
<td></td>
<td>particles.</td>
</tr>
<tr>
<td>Glow Map</td>
<td>(GPU only) Opens the Asset Browser to specify a glow map for the 2D GPU</td>
</tr>
<tr>
<td></td>
<td>particles.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>You must specify a positive value for the Emissive Lighting (p. 1529)</td>
</tr>
<tr>
<td></td>
<td>parameter for the glow map to be visible.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Texture Tiling</strong></td>
<td>Splits the texture into tiles for variation and animation.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Tiles X, Y</strong> – The number of tiles that the texture is split into.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 256</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td>• <strong>First Tile</strong> – The first of the range of tiles used by the particle.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 255</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td>• <strong>Variant Count</strong> – The number of consecutive tiles in the texture from which the particle randomly selects.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 256</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td>• <strong>Anims Frame Count</strong> – The number of tiles that compose an animation sequence. You can use <strong>Variant Count</strong> and <strong>Anim Frames Count</strong> together. For example, if <strong>Variant Count</strong> = 2 and <strong>Anim Frames Count</strong> = 8, the particle randomly chooses between tiles 0 to 7 or 8 to 15, as an animated sequence.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 to 256</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td>• <strong>Anim Framerate</strong> – The number of frames per second for the animation. If 0, the animation runs through one sequence in the particle lifetime.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td>• <strong>Anim Cycle</strong> – This parameter has the following values.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Once</strong> – Animation plays once and holds on the last frame</td>
</tr>
<tr>
<td></td>
<td>• <strong>Loop</strong> – Animation loops indefinitely</td>
</tr>
<tr>
<td></td>
<td>• <strong>Mirror</strong> – Animation alternates cycling forward and backward indefinitely</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>Once</strong></td>
</tr>
<tr>
<td></td>
<td>• <strong>Anim Blend</strong> – Renders the particle blended between the two adjacent animation frames. This has a performance impact.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td></td>
<td>• <strong>Horizontal Flip Chance</strong> – Specifies the fraction of particles that are rendered and mirrored in texture X.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Vertical Flip Chance</strong> – Specifies the fraction of particles that are rendered and mirrored in texture Y.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 to 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td>• <strong>Anim Curve</strong> – Modifies the speed and direction of the animation.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Color</td>
<td>Selects the color and alpha to apply to a particle.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Random Between Two Colors</strong> – Right-click the color input and choose the option to randomize between two colors and alpha values. Right-click to choose and return to <strong>Single Color</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Random</strong> – Specifies how much a particle’s initial color varies from the default.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 = no variation</td>
</tr>
<tr>
<td></td>
<td>• <strong>Random Hue</strong> – Causes the <strong>Random</strong> color variation to occur separately in the three color channels. If false, the variation is in luminance only.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td></td>
<td>• <strong>Strength Over Emitter Lifetime</strong> – Defines the color of the particle over the emitter’s lifetime. Double-click to open the <strong>Gradient Editor</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Strength Over Particle Lifetime</strong> – Defines the color of the particle over the particle’s lifetime. Double-click to open the <strong>Gradient Editor</strong>.</td>
</tr>
</tbody>
</table>
### Using the Particle Editor

#### Name: Alpha clip

*(CPU only)* Customizes how the particle **Alpha** value controls opacity or alpha test values. Each parameter has two values that correspond when particle **Alpha** = 0 and 1. They are interpolated for each particle with its **Alpha** value and used in the shader with the following equation:

\[
\text{FinalOpacity} = \text{saturate} \left( \frac{(\text{TextureAlpha} - \text{SourceMin})}{\text{min}(\text{SourceWidth}, 1 - \text{SourceMin})} \right) \cdot \text{Scale}
\]

- **Scale** – Multiplies the final alpha value. When set to the default value (0, 1), the particle **Alpha** directly scales the final opacity.
  - Valid values: 0+

- **Source Min** – Specifies the minimum source (texture) alpha to render (alpha test); values below the minimum become transparent. When set to the default value (0, 0), there is no alpha test.
  - Valid values: 0+

- **Source Width** – Specifies the feathering range of alpha clipping. When set to the default value (1, 1), texture alpha is fully used. A value of 0 specifies hard clipping. A value of 1 specifies soft clipping.
  - Valid values: 0+

  Default value: 0

  - Default – Alpha controls opacity with no alpha clipping: Scale = (0, 1), Source min = (0, 0), Source width = (1, 1).
  - Hard clipping at texture alpha = C, with no feathering: Scale = (1, 1), Source min = (C, C), Source width = (0, 0).
  - Hard clipping controlled by particle alpha: Scale = (1, 1), Source min = (0, 1), Source width = (0, 0).
  - Feathered clipping with width F, controlled by particle alpha: Scale = (1, 1), Source min = (0, 1), Source width = (F, F).
  - Soft clipping with the test value controlled by particle alpha: Scale = (1, 1), Source min = (0, 1), Source width = (1, 1)
  - Clipping and opacity scale controlled by particle alpha: Scale = (0, 1), Source min = (0, 1), Source width = (1, 1)

#### Tessellation

Enables tessellation, rendering more vertices in the sprite. You must have a minimum of DirectX 11. This parameter is useful when **Receive Shadows** is set, increasing the resolution of shadows. It is also useful when **Tail Length** or **Connection** are set, creating smoother curves in connected particles. This produces more accurate lighting when receiving light from point lights.

Default value: false
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Soft Particles     | Applies rendering that softens the intersection between sprites and nearby objects to prevent unnatural seams. Use sparingly on particles that need it, such as smoke, because this is slightly more expensive. Use the **Softness** parameter to define the amount of rendering to apply.  
| Default value: false |
| Motion Blur        | (GPU only) Simulates motion blur on GPU particles. Use **Blur Strength** to set the strength of the blur effect. |
| Geometry           | (CPU only) Opens the **Asset Browser** window to select a 3D object to use for the particles.  
| Default value: empty |
| Geometry in Pieces | (CPU only) Emits the geometry in pieces, originating at each piece's location in the asset. This applies only if the **Geometry** asset contains multiple subobjects.  
  - **Whole** – Emits the whole geometry asset as one object.  
  - **Random Pieces** – Emits geometry fragments randomly over the emitter lifetime.  
  - **All Pieces** – Emits geometry pieces in order over the emitter lifetime.  
| Default value: **Whole** |

This parameter requires that you provide specific names for the nodes in the Maya outliner. The following show examples of acceptable node names:

In these examples, the groups and individual nodes have a _group suffix. As a result, the Maya exporter assumes there is no geometry. If you receive a "group has no geometry" error, you can safely ignore it.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry No Offset</td>
<td>(CPU only) Uses the geometry pivot for centering geometry particles.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>Octagonal Shape</td>
<td>(CPU only) Renders sprites as octagons instead of quads, reducing pixel cost.</td>
</tr>
<tr>
<td></td>
<td>Only use with textures that fit within an octagon; otherwise clipping occurs.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
</tbody>
</table>

### Rotation Attribute

In the **Rotation** attribute, specify how to control the particle's rotation.

![Rotation Attribute](image)

### Rotation Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init Angles</td>
<td>XYZ values define the initial angle, in degrees, that is applied to the</td>
</tr>
<tr>
<td></td>
<td>particles upon spawning. For <strong>Facing = Camera</strong> particles, only the y-</td>
</tr>
<tr>
<td></td>
<td>axis is used and refers to rotation in screen space. For 3D particles, all</td>
</tr>
<tr>
<td></td>
<td>three axes are used and refer to emitter local space.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Random Angles</td>
<td>XYZ values define the random variation (bidirectional), in degrees, to</td>
</tr>
<tr>
<td></td>
<td><strong>Init Angles</strong>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Rotation Rate X, Y, Z</td>
<td>Specifies the constant particle rotation in degrees/second. The axes are</td>
</tr>
<tr>
<td></td>
<td>the same as <strong>Init Angles</strong>. You can set a <strong>Random</strong> value and **Strength</td>
</tr>
<tr>
<td></td>
<td>Over Emitter Lifetime** and <strong>Strength Over Particle Lifetime</strong> curves.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>
Size Attribute

In the **Size** attribute, specify how to control the particle’s size and shape.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lock Aspect Ratio</strong></td>
<td>Maintains the particle aspect ratio.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td><strong>Size X, Y, Z</strong></td>
<td>Specifies the world particle size.</td>
</tr>
<tr>
<td></td>
<td>For 2D sprite particles, only X and Y are used.</td>
</tr>
<tr>
<td></td>
<td>For geometry particles, X, Y, and Z are used. For more information, see the Geometry (p. 1539) parameter.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td><strong>Pivot X, Y, Z</strong></td>
<td>Moves the sprite's pivot point.</td>
</tr>
<tr>
<td></td>
<td>For 2D sprite particles, only X and Y are used.</td>
</tr>
<tr>
<td></td>
<td>For geometry particles, X, Y, and Z are used. For more information, see the Geometry (p. 1539) parameter.</td>
</tr>
<tr>
<td></td>
<td>Valid values: −1 to +1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0 (texture center)</td>
</tr>
<tr>
<td><strong>Stretch</strong></td>
<td>Specifies the amount of stretch, in seconds, that is applied to the particle in the direction of travel. This is based on the current velocity and stretches in both directions by default.</td>
</tr>
<tr>
<td></td>
<td><strong>Offset Ratio</strong> – Adjusts the center of stretching. 0 = stretch both directions</td>
</tr>
</tbody>
</table>
### Using the Particle Editor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Tail Length</strong></td>
<td>(CPU only) The length of the particle's tail in seconds. The particle texture is stretched through the tail.</td>
</tr>
<tr>
<td></td>
<td><strong>Tail Steps</strong> – The number of segments for the tail. A higher number produces smoother tail curves for nonlinear, moving particles.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Min Pixels</strong></td>
<td>(CPU only) Adds the specified number of pixels to the particle's true size when rendering. This is useful for important effects that should be visible even at a distance.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>

### Visibility Attribute

In the **Visibility** attribute, specify how to control the particle's visibility.

![Visibility Attribute](image)

- **Camera Non Facing Fade**
- **Fade Curve**
- **View Distance Adjust**
- **Camera Min Distance**
- **Camera Max Distance**
- **Camera Distance Offset**
- **Sort Offset**
- **Sort Bounds Scale**
- **Draw Near**
- **Draw On Top**
- **Visible Indoors**
- **Visible Underwater**
## Visibility Attribute Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Camera Non Facing Fade** | When enabled, this parameter adjusts the transparency of each particle based on its angle relative to the camera. You can specify this parameter to fade out particles at sharp angles, which hides the appearance of thin polygons. You can adjust the level of fade according to the angle with the **Fade Curve** tool.  
  Default value: false |
| **View Distance Adjust** | Multiplies the automatically computed fade-out camera distance.  
  Valid values: 0+  
  Default value: 1 |
| **Camera Min/Max Distance** | Determines the camera range that particles render in. A default value of 0 indicates an unlimited range.  
  Valid values: 0+  
  Default value: 0 |
| **Camera Distance Offset** | Offsets the emitter away from the camera.  
  Valid values: any  
  Default value: 0 |
| **Fade Strength Min/Max Distance** | (GPU only) Specifies the distance from the camera at which particles fade.  
  - **Fade Strength Min Distance** – Minimum distance from the camera at which particles fade.  
  - **Fade Strength Max Distance** – Maximum distance from the camera at which particles fade.  
  Default value: 0, 0 |
| **Sort Offset** | Bias the distance used for sorting. You can customize the sort order in an emitter tree. By default, subemitters render in the order they are listed in the effect. A bias of 0.01 or greater overrides that order. You can use larger biases to adjust the sorting order, with respect to other transparent objects in the level.  
  Valid values: any  
  Default value: 0 |
| **Sort Bounds Scale** | Specifies the emitter point for sorting.  
  1 = bounds nearest | 0 = origin | -1 = bounds farthest  
  Valid values: any  
  Default value: 0 |
| **Draw Near** | Renders particles in a near first-person space (with weapons). |
Using the Particle Editor

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Default value: false</td>
<td></td>
</tr>
<tr>
<td>Draw on Top</td>
<td>Renders particles on top of everything (no depth test).</td>
</tr>
<tr>
<td>Default value: false</td>
<td></td>
</tr>
<tr>
<td>Visible Indoors</td>
<td>For use in vis areas.</td>
</tr>
<tr>
<td>• If_False</td>
<td>Hides particles when indoors.</td>
</tr>
<tr>
<td>• If_True</td>
<td>Hides particles when outdoors.</td>
</tr>
<tr>
<td>• Both</td>
<td>Shows particles when indoors or outdoors.</td>
</tr>
<tr>
<td>Default value: Both</td>
<td></td>
</tr>
<tr>
<td>Visible Underwater</td>
<td>For use with ocean and water volumes.</td>
</tr>
<tr>
<td>• If_False</td>
<td>Hides particles when under water.</td>
</tr>
<tr>
<td>• If_True</td>
<td>Hides particles when above water.</td>
</tr>
<tr>
<td>• Both</td>
<td>Shows particles when under or above water.</td>
</tr>
<tr>
<td>Default value: Both</td>
<td></td>
</tr>
</tbody>
</table>

Creating and Managing Particle Emitters

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create, edit, and manage emitters for particle effects in the Particle Editor. For more information about emitter attributes and parameters displayed in the Attributes panel, see Particles Attributes Reference (p. 1513).

Before you can create or edit emitters, you must first set up a particle library. For more information, see Adding Particle Libraries (p. 1508).

Creating Emitters

Do the following to create an emitter. You must have already created a library.

To create new emitters

1. In Lumberyard Editor, choose Tools, Particle Editor.
2. In the Libraries panel, under your selected library, click Add Particle. Alternatively, right-click the library name and then choose Add New, Add Particle.
3. Enter a name for the emitter and press Enter. Do not use special characters in the name.
4. In the Attributes panel, edit the attributes and parameters as needed.

Editing Emitters

Do the following to edit emitter attributes and parameters.
To edit emitter attributes and parameters

1. In Lumberyard Editor, click Tools, Particle Editor.
2. In the Libraries panel, choose an emitter.
3. In the Attributes panel, adjust the attribute and parameter settings and values to achieve the intended effect.

Duplicating Emitters

Do the following to duplicate an emitter. This will also duplicate any associated child emitters.

To duplicate emitters

1. In Lumberyard Editor, click Tools, Particle Editor.
2. In the Libraries panel, select the emitter that you want to duplicate. Right-click the emitter and choose Duplicate. Alternatively, press Ctrl+D.
3. In the dialog box, enter a name for the emitter and click OK.

Topics
- Creating Child Emitters (p. 1545)
- Organizing Emitters in a Library (p. 1546)
- Reverting Changes to Emitter Attributes (p. 1546)

Creating Child Emitters

To create a child emitter, you first set the parent effect and then attach the child emitter to the parent particles. You can attach multiple child emitters to the parent particle. A particle effect can have any number of child effects (also called subeffects), which you can nest in a library by dragging and dropping where needed.

To create child emitters

1. In Lumberyard Editor, choose Tools, Particle Editor.
2. In the Libraries panel, right-click the emitter for which you want to create a child emitter. Choose Add New, Add Particle.
3. Enter a name for the child emitter and then press Enter.

To assign an existing emitter to be a child

- Drag the emitter on top of another emitter. The selected emitter is nested underneath as a child.

To remove a child emitter from a parent

- Drag the child emitter to the preferred location. Alternatively, you can drag the child emitter to the library name to detach it from its parent and make it appear at the bottom of the emitter stack.

Lumberyard has two types of child effects:

- Regular child effects – These effects behave like separate effects, though they are spawned with and attached to their parent effect. Each child effect has its own independent parameters and lifetime, allowing for an overall effect that consists of several parts.
• **Second-generation child effects** – These effects are attached to individual particles of the parent effect. A separate emitter is spawned for each particle of the parent effect, and those emitters move with their parent particles. This allows you to create more complex effects. You can nest second-generation effects multiple times in order to create third-generation (and greater) effects.

An example of a child effect is attaching an emitter to a parent particle and leaving trailing particles behind. You can achieve this effect using the **Spawn Indirection** attribute.

**Organizing Emitters in a Library**

All particle emitters are listed in the Libraries panel. You can organize and create relationships between your emitters. For example, you can have single emitters or emitters with various child hierarchies. You can also create directories and groups in each library to organize your particle effects. This relationship is displayed in a tree hierarchy in the Libraries panel.

A visual indicator shows you the placement of an emitter based on the position of your cursor. For example, if you place an emitter on another emitter or a folder for grouping, the folder row appears highlighted with a blue stroke.

**Reverting Changes to Emitter Attributes**

Emitters have a list of attributes or property types that are categorized for easier identification. You can reorder and rearrange the categories, including combining categories into a tabbed view.

- To reorder categories, drag and drop the category to the preferred position. An orange highlight appears to indicate a valid docking location.
- To combine categories into tabs, drag a category on another category title bar. If you have not expanded the category, the tabs appear collapsed until you click a category title bar.
- To revert changes made to the Attributes panel layout, click the menu in the title bar and choose **Reset to default**.

By default, emitter parameters are set to the attribute as a common starting point. These default attributes have a white text label. When you change the attribute parameter from the default state, the text label changes from white to orange.

- To revert the last change to the emitter attribute, click **Edit, Undo**. Alternatively, you can press Ctrl+Z.
- To revert all changes to the attributes parameter, right-click the attribute name and choose **Reset to default**.

**Working with Emitter Shape Types**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Particle Editor provides emitter shape types that you can use to achieve a variety of effects, such as using the **Beam** emitter to create fire effects.

**To choose the emitter shape type**

1. In Lumberyard Editor, choose **Tools, Particle Editor**.
2. In the Libraries panel, choose an emitter.
3. In the **Attributes** panel, choose **Emitter**.
4. For **Emitter Shape Type**, choose the shape type. The following emitter shape types are supported:
   - **Angle**
   - **Beam**
   - **Box**
   - **Circle**
   - **Point**
   - **Sphere**
   - **Trail**

Most emitters have CPU and GPU particle types. Each emitter's velocity functions on either world or shape axial coordinates, based on type. To view the shape XYZ coordinates, right-click the **Preview** viewport and choose **Show Emitter Shape**.

**Topics**

- **Angle Emitter (CPU and GPU)** (p. 1547)
- **Beam Emitter (CPU)** (p. 1550)
- **Box Emitter (CPU and GPU)** (p. 1553)
- **Circle Emitter (CPU and GPU)** (p. 1554)
- **Point Emitter (CPU and GPU)** (p. 1556)
- **Sphere Emitter (CPU and GPU)** (p. 1558)
- **Trail Emitter (CPU)** (p. 1560)

**Angle Emitter (CPU and GPU)**

The **Angle** emitter spawns particles in a directional pattern based on the parameters. This is done to control the angle of emission from its source.

The following are parameters for the **Angle** emitter (CPU).
## Angle Emitter Parameters for CPU Attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Focus Angle</strong></td>
<td>Specifies the number of degrees to rotate from the y-axis.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 (up) – 180 (down)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Focus Azimuth</strong></td>
<td>Specifies the number of degrees to rotate the new axis about the y-axis.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any (0, 360 = North, 90 = West, 180 = South, 270 = East)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Focus Camera Direction</strong></td>
<td>Sets the focus direction to face the camera. You can set a Random value and the Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 – 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Focus Gravity Direction</strong></td>
<td>Ignores the emitter rotation and focuses gravity on world Z.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td><strong>Focus Rotates Emitter</strong></td>
<td>Default value: false</td>
</tr>
<tr>
<td><strong>Emit Offset Direction</strong></td>
<td>If true, changes each particle's emission direction to align with its offset from the origin.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td><strong>Emit Angle</strong></td>
<td>Sets the angle deviation of an emitted particle from the default focus (+Y) axis. 0 = up, 90 = horizontal, 180 = down. This is the maximum angle from the focus. You can set a Random value (determines minimum angle) and the Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>• To emit in all directions, set Emit Angle = 180, Random = 1.</td>
</tr>
<tr>
<td></td>
<td>• To emit in the top hemisphere, set Emit Angle = 90, Random = 1.</td>
</tr>
<tr>
<td></td>
<td>• To emit in a horizontal circle, set Emit Angle = 90, Random = 0.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Curvature** | Sets how far to bend the vertex normals for **Facing=Camera** particles into a spherical shape. This affects lighting.  
Valid values: 0 (flat) – 1 (hemispherical shape)  
Default value: 1 |

### Angle Emitter Parameters for GPU Attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Focus Angle** | Specifies the number of degrees to rotate from the y-axis  
Valid values: 0 (up) – 180 (down)  
Default value: 0 |
| **Focus Azimuth** | Specifies the number of degrees to rotate the new axis about the y-axis.  
Valid values: any (0, 360 = North, 90 = West, 180 = South, 270 = East)  
Default value: 0 |
| **Emit Angle** | Sets the angle deviation of an emitted particle from the default focus (+Y) axis. 0 = up, 90 = horizontal, 180 = down. This is the maximum angle from the focus. You can set a Random value (determines minimum angle) and the Strength over Emitter Lifetime curve  
Valid values: 0 – 180  
Default value: 0 |

- To emit in all directions, set **Emit Angle** = 180, **Random** = 1.  
- To emit in the top hemisphere, set **Emit Angle** = 90, **Random** = 1.  
- To emit in a horizontal circle, set **Emit Angle** = 90, **Random** = 0.
Beam Emitter (CPU)

The **Beam** emitter spawns a length of connected particles at once, based on the origin and specified target position. You can also set wave form attributes to shape or animate the beam.

The following are parameters for the **Beam** emitter (CPU).

### Beam Emitter Parameters for CPU Attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Beam Lifetime</strong></td>
<td>Sets the beam lifetime, in seconds. You must use this in conjunction with particle lifetime.  &lt;br&gt;Valid values: 0+  &lt;br&gt;Default value: 1</td>
</tr>
<tr>
<td><strong>Target Position</strong></td>
<td>Specifies the XYZ position to set the beam target, offset from the origin.  &lt;br&gt;Default value: 0, 0, 15</td>
</tr>
<tr>
<td><strong>Random Target Offset</strong></td>
<td>Randomizes the XYZ beam target, offset from the target position.  &lt;br&gt;Valid values: any  &lt;br&gt;Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Segment Type</strong></td>
<td>Specifies how each segment generates over the length of the beam. Use <strong>Fixed</strong> to set the number of segments over the length. Use <strong>Length</strong> to set the length of each segment over the beam length.  &lt;br&gt;Valid values: <strong>Fixed</strong>, <strong>Length</strong></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>Fixed</strong></td>
</tr>
<tr>
<td><strong>Segment Count</strong></td>
<td>Defines the number of segments with the <strong>Fixed</strong> type set.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td><strong>Segment Length</strong></td>
<td>Defines the length of each segment with the <strong>Length</strong> type set.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td><strong>Texture Shift</strong></td>
<td>Shifts the texture coordinate in the V direction at the specified rate.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Up Vector for Waves</strong></td>
<td>Specifies the XYZ vector on which the waveform occurs.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Wave Form Source</strong></td>
<td>Specifies the side of the beam from which to generate the waveform.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Origin</strong> – Waveform is calculated from the emitter origin.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Target</strong> – Waveform is calculated from the target specified by the target attributes.</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>Origin</strong></td>
</tr>
<tr>
<td><strong>Wave Form Type</strong></td>
<td>Sets the behavior for the waveform.</td>
</tr>
<tr>
<td></td>
<td>• <strong>None</strong> – No waveform is used.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Sine</strong> – A periodic waveform generates a smooth, repetitive curve.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Square</strong> – A nonsinusoidal, periodic waveform in which the amplitude alternates.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Noise</strong> – A repetitive waveform with a randomized amplitude.</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>None</strong></td>
</tr>
</tbody>
</table>
### Using the Particle Editor

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Wave Phase**   | Sets the position of the waveform at the specified source. A value of 360 completes one full cycle of the waveform. You can animate the beam waveform by using **Emitter Lifetime**, **Pulse Period**, and **Strength over Emitter Lifetime** in conjunction.  

**Note**  
The beam animation is updated when a new one is spawned based on the **Beam Lifetime** parameter. A lower lifetime provides a smoother animation.  

The following example demonstrates the wave phase parameter:  
Beam lifetime = .01, Emitter lifetime = 2, Pulse period = 2, Up vector = 1, 0, 0, Wave form type = Sine, wave phase = 180 (with a curve set on **Strength over Emitter Lifetime**), Wave amplitude = 2, Wave frequency = 0.5  
Valid values: 0+  
Default value: 0  

<table>
<thead>
<tr>
<th><strong>Wave Amplitude</strong></th>
<th>Sets the strength of the waveform deformation.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Wave Frequency</strong></th>
<th>Sets the number of wave cycles over the waveform length.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Texture Mapping</strong></th>
<th>Maps the assigned texture to each particle quad or across the trail stream.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• <strong>Per Particle</strong> – Texture is mapped on each quad.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Per Stream</strong> – Texture is mapped over the length of the beam.</td>
</tr>
<tr>
<td>Default value:</td>
<td><strong>Per Particle</strong></td>
</tr>
</tbody>
</table>

**Note**  
When there is a single light source, some or all particles for **Trail** emitters can appear unlit (black). This is most apparent when the sun is the only light source and the time and day setting is noon. To address this:  

1. Add an Environment Probe to the scene to create indirect lighting. For more information, see **Environment Probe** (p. 587).  
2. Enable Environment Probe Lighting on the particle system by specifying a value for the Environment Probe Lighting parameter. For example, a value of 0.5 applies light from the Environment Probe at half intensity. For more information, see **Lighting Attribute** (p. 1528).
Box Emitter (CPU and GPU)

The **Box** emitter enables spawning of particles along the shapes axial coordinate system. This allows you to create complex planar and rectangular effects.

The following is an example of the **Box** emitter in the **Preview** viewport.

The following are parameters for the **Box** emitter.

### Shape Parameters
- Emitter Size
- Confine X
- Confine Y
- Confine Z
- Spawn Pos XYZ
- Spawn Pos XYZ Random
- Velocity XYZ
- Velocity XYZ Random

### Box Emitter Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emitter Size XYZ</td>
<td>Sets the size of the box emitter in meters.</td>
</tr>
</tbody>
</table>
### Parameter | Description
--- | ---
**Confine XYZ** | Confines the particles to render within the box size on each axis.  
Valid values: 0+ (radius)  
Default value: 5

**Spawn Pos XYZ** | XYZ values define the spawning position away from the emitter in emitter space. You can set a Random value and Strength over Emitter Lifetime curve.  
Valid values: any  
Default value: 0, 0, 0

**Spawn Pos XYZ Random** | For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a Random value and Strength over Emitter Lifetime curve.  
Valid values: any  
Default value: 0, 0, 0

**Velocity XYZ** | XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis in which velocity spread occurs around. You can set a Random value and Strength over Emitter Lifetime curve.  
Valid values: any  
Default values: 0, 0, 0

**Velocity XYZ Random** | For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a Random value and Strength over Emitter Lifetime curve.  
Valid values: any  
Default value: 0, 0, 0

---

**Circle Emitter (CPU and GPU)**

The Circle emitter enables spawning of particles along the shapes axial coordinate system (see diagram below). This allows you to create complex circular and cylindrical effects.

The following is an example of the Circle emitter in the Preview viewport.
The following are parameters for the Circle emitter.

**Shape Parameters**
- Emitter Size
- Spawn Pos XYZ
- Spawn Pos XYZ Random
- Spawn Pos Increment XYZ
- Spawn Pos Increment XYZ Random
- Velocity XYZ
- Velocity XYZ Random

**Circle Emitter Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emitter Size</td>
<td>Sets the size of the circle emitter in meters.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+ (radius)</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Spawn Pos XYZ</strong></td>
<td>XYZ values define the spawning position away from the emitter itself in emitter space. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Spawn Pos XYZ Random</strong></td>
<td>For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Spawn Pos Increment XYZ</strong></td>
<td>Spawns each subsequent particle at incremental positions along the axis over the emitter size, based on a percentage value. For example, if you set a value of 20, a particle spawn every 20% along the axis, making 5 particles on the axis to equal 100%. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any (percentage)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Spawn Pos Increment XYZ Random</strong></td>
<td>Provides another layer of randomization for each particle to spawn at percentages over the axis, independent of the Spawn Pos Increment parameter. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any (percentage)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Velocity XYZ</strong></td>
<td>XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis around which velocity spread occurs. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default values: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Velocity XYZ Random</strong></td>
<td>For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
</tbody>
</table>

**Point Emitter (CPU and GPU)**

The Point emitter spawns particles in a spherical distribution from the origin. This allows you to control the spread angle to create complex conical and spherical effects.
The following are parameters for the **Point** emitter.

### Point Emitter Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spawn Offset</strong></td>
<td>Offsets the spawning of particles spherically at distance from the emitter origin. Valid values: 0+ (radius) Default value: 0</td>
</tr>
<tr>
<td><strong>Spawn Pos XYZ</strong></td>
<td>XYZ values define the spawning position away from the emitter itself in emitter space. You can set a <strong>Random</strong> value and <strong>Strength over Emitter Lifetime</strong> curve. Valid values: any Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Spawn Pos XYZ Random</strong></td>
<td>For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a Random value and Strength over Emitter Lifetime curve. Valid values: any Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Velocity XYZ</strong></td>
<td>XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis around which velocity spread occurs. You can set a Random value and Strength over Emitter Lifetime curve. Valid values: any Default values: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Velocity XYZ Random</strong></td>
<td>For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a Random value and Strength over Emitter Lifetime curve. Valid values: any Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Velocity Spread</strong></td>
<td>Restricts the angle of spherical distribution in the direction of the velocity XYZ vector. For example, a value of 360 = Sphere, 180 =</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Hemisphere, and so on. You can set a Random value and Strength over Emitter Lifetime curve.</td>
<td></td>
</tr>
<tr>
<td>Valid values: 0 – 360</td>
<td></td>
</tr>
<tr>
<td>Default value: 360</td>
<td></td>
</tr>
</tbody>
</table>

**Sphere Emitter (CPU and GPU)**

The **Sphere** emitter enables spawning of particles along the shapes axial coordinate system (see diagram below). This allows you to create complex spherical effects.

The following is an example of the **Sphere** emitter in the **Preview** viewport.

The following are parameters for the **Sphere** emitter.
**Sphere Emitter Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Emitter Size</strong></td>
<td>Sets the size of the sphere emitter in meters.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+ (radius)</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td><strong>Spawn Pos XYZ</strong></td>
<td>XYZ values define the spawning position away from the emitter itself in emitter space. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Spawn Pos XYZ Random</strong></td>
<td>For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
<tr>
<td><strong>Spawn Pos Increment XYZ</strong></td>
<td>Spawns each subsequent particle at incremental positions along the axis over the emitter size, based on a percentage value. For example, if you set a value of 20, a particle spawns every 20% along the axis, making 5 particles on the axis to equal 100%. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any (percentage)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>Spawn Pos Increment XYZ Random</strong></td>
<td>Provides another layer of randomization for each particle to spawn at percentages over the axis, independent of the Spawn Pos Increment parameter. You can set a Random value and Strength Over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any (percentage)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>
Using the Particle Editor

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity XYZ</td>
<td>XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis around which velocity spread occurs. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default values: 0, 0, 0</td>
</tr>
<tr>
<td>Velocity XYZ Random</td>
<td>For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a Random value and Strength over Emitter Lifetime curve.</td>
</tr>
<tr>
<td></td>
<td>Valid values: any</td>
</tr>
<tr>
<td></td>
<td>Default value: 0, 0, 0</td>
</tr>
</tbody>
</table>

**Trail Emitter (CPU)**

The Trail emitter connects particles together to create a trailing effect as the system moves through space.

The following are parameters for the Trail emitter.

**Trail Emitter Parameters**

<table>
<thead>
<tr>
<th>Parameter Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect to Origin</td>
<td>Connects a newly spawned particle to the origin.</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>Texture Mirror</td>
<td>Mirrors alternating texture tiles.</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>Texture Frequency</td>
<td>Specifies the number of texture wraps per trail emitter sequence, based on the Texture Mapping type.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0+</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
</tbody>
</table>
Using the Particle Editor

Parameter Function | Description
--- | ---
**Lock Anchor Points** | Locks the UV anchor points of the texture to stay in place, rather than follow the emitter location.

- Default value: false

**Texture Mapping** | Maps the assigned texture to each particle quad or across the trail stream.

- **Per Particle** – Texture is mapped on each quad.
- **Per Stream** – Texture is mapped over the length of the beam.

- Default value: **Per Particle**

**Note**

When there is a single light source, some or all particles for Trail emitters can appear unlit (black). This is most apparent when the sun is the only light source and the time and day setting is noon. To address this:

1. Add an Environment Probe to the scene to create indirect lighting. For more information, see Environment Probe (p. 587).
2. Enable Environment Probe Lighting on the particle system by specifying a value for the Environment Probe Lighting parameter. For example, a value of 0.5 applies light from the Environment Probe at half intensity. For more information, see Lighting Attribute (p. 1528).

**Particle Trail Visibility**

Trail segments are drawn when the distance between the start and end of a segment exceed the value for **Min visible distance**. You can choose to automatically disable the drawing of trails that are not moving or are moving too slowly. This is useful if you have particle trail effects that need to be drawn only when the emitter is moving.

The following are movement parameters for the Trail emitter.

**Related Movement Attributes**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min visible segment length</strong></td>
<td><em>(Trail Emitter (CPU) (p. 1560) only)</em> Tail particles are visible only when they have moved the specified distance.</td>
</tr>
</tbody>
</table>

- Default value: false

| **Min visible distance** | *(Trail Emitter (CPU) (p. 1560) only)* Specifies the minimum distance between the start and end of a trail segment. Segments that are smaller than this value become transparent. |

- Valid values: 0+
Creating Custom Attribute Panels

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Create a custom particle attribute panel so you can quickly access the parameters that you frequently use. You can drag parameters from an existing standard panel to copy the parameters into the custom panel. Or you can drag parameters from a custom panel to move the parameters to the new custom panel. If you want to simultaneously add multiple parameters, press Ctrl while selecting the parameters.

When you move a panel to a new location, an indicator appears so you can preview a valid drop location. If you do not see an indicator, the parameter is inserted at the end of the panel. The following example shows a custom panel with drop indicators.
To create a custom attribute panel

1. In Lumberyard Editor, choose **Tools, Particle Editor**.
2. In the **Particle Editor**, click the menu in the **Attributes** title bar.
3. Do one of the following:
   - Choose Custom attributes, New attribute.
   - Choose Custom attributes, Import attribute.

4. Name your custom attribute or browse to the attribute that you want to import.

5. Once created, you can right-click the attribute title bar to rename, empty, or export custom attribute panels.

Custom Attributes Panel Menu Options

Use the following menu options to modify your custom attribute panel.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New attribute</td>
<td>Adds an empty, custom panel to the Attributes panel.</td>
</tr>
<tr>
<td>Import attribute</td>
<td>Loads an existing custom attribute panel. The panel is also added to the panel preset list.</td>
</tr>
<tr>
<td>Panel preset list</td>
<td>Lists the custom panel presets.</td>
</tr>
<tr>
<td>Reset list</td>
<td>Resets the preset list.</td>
</tr>
<tr>
<td>Rename</td>
<td>Renames the custom panel.</td>
</tr>
<tr>
<td>Remove all</td>
<td>Removes all attributes in the panel.</td>
</tr>
</tbody>
</table>
Option | Description
---|---
Export | Exports the selected custom panel as a .custom_attribute file. The panel is also added to the panel preset list.
Close | Closes the custom panel.

### Using the Color Picker

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the color picker to apply a color tint and alpha (opacity) to your particles. You can save your palette libraries to share with others.

**To use the Color Picker**

1. In Lumberyard Editor, choose Tools, Particle Editor.
2. In the Libraries panel, select an emitter.
3. In the Attributes panel, under Particles, click the color swatch for the Color parameter.
The **Color Picker** includes the following:

A. **Eye dropper** – Selects a color on your screen with magnification.
B. **Current** and **New color** – Displays your current color selection and the new color selection.
C. HTML and RGB values – Specifies the color value inputs.
D. Color window – View the range of colors to select.
E. Channels – Focuses the hue range based on RGB and CMY color sets.
F. Hue, Saturation, Value, and Alpha – Slides to select a new color.
G. Default Library palette – A list or grid of the currently selected library palette.
H. Help icon – Displays the online help topic for the Color Picker.

Using the Palette Libraries

Create a palette library to save your collection of custom colors. You can create multiple palette libraries.

Use the following actions in the palette library menu.
Using the Particle Editor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>Displays a list view with name or RGBA values.</td>
</tr>
<tr>
<td>Grid</td>
<td>Displays the palette as a grid of color swatches.</td>
</tr>
<tr>
<td>Preset preview size</td>
<td>Changes the palette swatch size.</td>
</tr>
<tr>
<td>Create New Library</td>
<td>Creates a blank palette library.</td>
</tr>
<tr>
<td>Libraries</td>
<td>Displays a list of imported palette libraries.</td>
</tr>
<tr>
<td>Load Library</td>
<td>Imports a saved palette library.</td>
</tr>
<tr>
<td>Export</td>
<td>Exports the current palette library to a file.</td>
</tr>
<tr>
<td>Rename Library</td>
<td>Renames the current palette library.</td>
</tr>
<tr>
<td>Remove Library</td>
<td>Removes the current palette library.</td>
</tr>
<tr>
<td>Reset to default</td>
<td>Resets the current library to the default palette.</td>
</tr>
<tr>
<td>Remove All Libraries</td>
<td>Removes all palette libraries from the libraries list. This does not delete exported library palettes.</td>
</tr>
</tbody>
</table>

Using the Gradient Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the **Gradient Editor** to apply color ranges to an emitter and configure additional subparameters.

**To use the Gradient Editor**

1. In Lumberyard Editor, choose **Tools, Particle Editor**.
2. In the **Libraries** panel, select an emitter.
3. In the **Attributes** panel, under **Particles**, expand **Color**.
4. Click the color swatch for the **Strength Over Emitter Lifetime** or **Strength Over Particle Lifetime** parameter.
Gradient Editor UI Options

The Gradient Editor includes the following:

- **Location** – Sets the location value range from 0% – 100%.
- **Color** – Opens the Color Picker.
- **Gradient box** – Applies the combined gradient and alpha.
- Gradient viewport
  - The x-axis represents the gradient generator of the color change over the full gradient.
  - The y-axis represents 0 – 100% alpha of the gradient color.
- **Default Library**, alpha curve – Provides alpha curves to use as a starting point.
- **Default Library**, gradient – Provides gradients to use as a starting point.

Working with Color Gradients

When you select a gradient from the Default Library, it displays with the alpha curve in the Gradient Editor viewport. You can do the following when selecting a gradient:

**To change the gradient color**

- Click the triangle keyframes and select a new color from the Color Picker.
To add a color to the gradient

- Double-click the X-axis to generate a color keyframe. The color in the color thumbnail is added to the gradient viewport. Any adjustments you make to the gradient is reflected in the gradient viewport.

To display the RGBA values

- Pause on the color keyframe.

To delete a color keyframe

- Select the keyframe and press Delete. The selected keyframe has an orange outline.

To adjust the alpha curve

- Click and drag the circle (alpha keyframe) in the gradient viewport. Move the circle up (towards 100%) or down (towards 0%) to adjust the alpha percentage. Moving the circle left or right adjusts the curve based on the curve endpoints.

To display the alpha curve context menu

- Right-click the alpha curve keyframe. The following options are available: Delete selected keys, Create flat or linear curves, Adjust the in-and-out tangent of the curve to be linear or flat, Add a created curve to the library or preset list, and Reset the curve to defaults.

To add an alpha key

- Double-click the curve in the gradient viewport.

To delete an alpha key

- Select the circle key and press Delete.

To add a generated alpha curve to the preset list

- Click the + button.

To add a generated gradient to the gradient preset list

- Click the + button.

To delete a curve or gradient preset

- Right-click the gradient or curve and click Remove.

Managing Particle Level of Detail (LOD)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The Level of Detail (LOD) system blends multiple particle emitters based on their distance from the camera. This allows you to use emitters that require less computation and rendering time, rather than computationally heavy particle emitters.

**To add an LOD for a particle emitter**

1. In Lumberyard Editor, choose Tools, Particle Editor.
2. In the Particle Editor, in the Libraries panel, right-click an emitter.
3. Select Add LOD.

The LOD is a copy of the base particle emitter and has the same settings. The LOD also applies to all parent or child particle emitters in the hierarchy that belong to the selected emitter.

**Level of Detail Panel**

The Level of Detail panel appears when you add an LOD. This panel shows the level of detail that you selected from the View menu in the Particle Editor.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blend In</td>
<td>The amount of time, in seconds, for an LOD to blend in.</td>
</tr>
<tr>
<td>Blend Out</td>
<td>The amount of time, in seconds, for an LOD to blend out.</td>
</tr>
<tr>
<td>Overlap</td>
<td>The amount of time, in seconds, that both LODs are shown before the old LOD blends out and the new LOD blends in.</td>
</tr>
</tbody>
</table>
## Using the Particle Editor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Add Level of Detail</td>
<td>Adds an LOD. The new LOD distance is set to 10 additional units from the farthest LOD. The list of added LOD levels appears under + Add Level of Detail.</td>
</tr>
</tbody>
</table>

### Hide or Display Options

Hide or display the **Level of Detail** panel by clicking **View, Hide Level of Detail** in the **Particle Editor**.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide Level of Detail</td>
<td>Hides the LOD panel if the panel is visible.</td>
</tr>
<tr>
<td>Show Level of Detail</td>
<td>Displays the LOD panel if the panel is hidden.</td>
</tr>
</tbody>
</table>

### Manage LOD Options

Manage your LOD levels in the list by clicking the drop-down menu in the **Level of Detail** title bar.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add level</td>
<td></td>
</tr>
<tr>
<td>Arrange</td>
<td></td>
</tr>
<tr>
<td>Jump to first</td>
<td></td>
</tr>
<tr>
<td>Jump to Last</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td></td>
</tr>
<tr>
<td>Remove all</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
</tr>
</tbody>
</table>
Using the Particle Editor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add level</td>
<td>Adds an LOD level to the end of the list.</td>
</tr>
<tr>
<td>Arrange</td>
<td>Shows the Arrange submenu:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Move up</strong> – Moves the selected LOD level one position up in the list.</td>
</tr>
<tr>
<td></td>
<td>This also changes the <strong>Level LOD Distance</strong> to 1.0 lower than the previous level.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Move down</strong> – Moves the selected LOD level one position down in the list.</td>
</tr>
<tr>
<td></td>
<td>This also changes the <strong>Level LOD Distance</strong> to 1.0 higher than the previous level.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Move to top</strong> – Moves the selected LOD level to the top of the list.</td>
</tr>
<tr>
<td></td>
<td>This also changes the <strong>Level LOD Distance</strong> to 1.0 lower than the previous top level.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Move to bottom</strong> – Moves the selected LOD level to the end of the list.</td>
</tr>
<tr>
<td></td>
<td>This also changes the <strong>Level LOD Distance</strong> to 1.0 higher than the previous bottom level.</td>
</tr>
<tr>
<td>Jump to first</td>
<td>Selects the first LOD level in the list.</td>
</tr>
<tr>
<td></td>
<td>This also selects the top particle emitter in the list and loads it in the Attributes panel.</td>
</tr>
<tr>
<td>Jump to last</td>
<td>Selects the last LOD level in the list.</td>
</tr>
<tr>
<td></td>
<td>This also selects the top particle emitter in the list and loads it in the Attributes panel.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected LOD level.</td>
</tr>
<tr>
<td>Remove All</td>
<td>Removes all LOD levels for the related particle emitters.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the <strong>Level of Detail</strong> panel.</td>
</tr>
</tbody>
</table>

**LOD Level Panel**

Each LOD level has its own panel in the LOD level list. These panels show all relevant information for each individual level.

**Note**

The base particle emitter is shown if the camera distance is lower than the top LOD level. This makes the base particle the starting LOD level.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top left check box</td>
<td>Turns on/off the entire level.</td>
</tr>
<tr>
<td>LOD distance value</td>
<td>Specifies the camera distance at which the LOD level becomes active. This level, the particle emitter blends toward the LOD level and blends out the previous LOD level.</td>
</tr>
<tr>
<td>Top right button</td>
<td>Deletes the corresponding LOD level.</td>
</tr>
</tbody>
</table>
### Using the Particle Editor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle check box</td>
<td>Turns off the particle emitter at this level. When the emitter is off, nothing is drawn. You can use this to turn off particle emitters based on the LOD level.</td>
</tr>
<tr>
<td>Particle name</td>
<td>Click a particle name to load the LOD level particle emitter for the selected particle emitter in the Attributes panel. You can use this to change the LOD level particle emitter. Right-click a particle name and click Remove to remove the particle from the LOD level. Any child particle emitters are also removed from the LOD level.</td>
</tr>
</tbody>
</table>

#### Using Particle Editor Keyboard Shortcuts

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Most of the commands in the Particle Editor menus have corresponding keyboard shortcuts. You can edit the keyboard shortcuts by clicking Edit, Edit Hotkeys. Modify as needed in the HotKey Configuration window.

**Hotkey Configuration**

**Export**

Exports the keyboard shortcuts (hotkey) list to a file.

**Import**

Imports a keyboard shortcuts (hotkey) list from a file.

**Reset to Default**

Resets all keyboard shortcuts (hotkeys) to the editor default settings.

**Click to assign**

Records a new keyboard shortcut when you click the shortcut. Clears the keyboard shortcut when you right-click the shortcut. Click OK to save your changes.

The Particle Editor uses the following keyboard shortcuts.
Particle Effects Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The total number of particles in a scene is not a critical factor when considering best practices for working with particles. Total fill-rate, physics, and (to some extent) spawn rate are more important. Following are some best practices for working with particles:

- Use soft particles only on subemitters that are near the ground and have only small particles. Create similar subemitters higher up that emit particles that never intersect with the ground and don't require soft particles.
- If sharp details are not required, use low-resolution textures and texture compression.
- Use an alpha texture with high or average opacity rather than additive blending.
- Each second-generation effect causes an emitter to be created for each particle in the parent effect. Use this sparingly because it can be expensive.
- Use physicalized particles sparingly because they are expensive. You can split an effect into subeffects, so that only a few large particles have physics enabled for appearance. The rest go through the ground or fade out quickly.
- Instead of multiple overlaid sprites for chaotic glow effects, use only two particles at a time. Carefully tune the lifetime and rotation rate, and set curves for Alpha, Color, and Size to combine in chaotic ways. Or, increase the Emissive Lighting parameter.
- For large, full-screen particles, use a Fill Rate Cost value of 1 or above.
- For small particles, such as sparks, set a maximum distance value to ensure that they aren't rendered as small, single pixel particles. Use the lowest Config spec setting to turn off small particles that are used in collisions.

Advanced Particle Techniques

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following advanced techniques in the Particle Editor to emit particles from geometry, create a particle effect that explodes, preview your effects on an animated character, or generate particles from surface properties.

Emitting Particles from Geometry

Do the following to emit particles from the parent geometry.

To emit particles from geometry

1. Create a parent emitter with a .cgf mesh asset that is assigned in the Geometry input under Particles. For more information, see Creating and Managing Particle Emitters (p. 1544).
2. Create a new emitter as a child of the first emitter.
3. On the child emitter, in the emitter attributes, set **Spawn Indirection** to **Parent Start**.
4. Set the **AttachType** to **Render**, and the **AttachForm** to **Vertices**.

The child's particles spawn from the mesh of its parent.

![Advanced Particle Techniques](image)

**Note**

Use any **Attach Type** and **AttachForm** for the intended effect.

**Creating Exploding Geometry Particle Effect**

Do the following to create a particle effect that instantly spawns exploding chunks.

**To create an exploding geometry particle effect**

1. Create an emitter. In the **Attributes** panel, under **Particles**, assign a multipart .cgf file to the **Geometry** input.
2. Set **Geometry** in **Pieces** to the option for your use case: **RandomPieces** or **AllPieces**.
3. Set appropriate values for **Gravity**, **Speed**, **Rotation Rate**, and more to create an exploding effect.
4. Optionally set **Collision** parameters for physicalized pieces.

**Attaching Particles to Character Animations for Previewing**

Do the following to preview the effects on an animated character in your level.

**To attach a particle to an animation**

1. In Lumberyard Editor, choose **Tools, Asset Browser**.
2. In the **Asset Browser**, select and drag a .cdf character asset that has animations into the Lumberyard Editor viewport. Close the **Asset Browser**.
3. In the **Entity Inspector**, click **Add Component**. Under **Animation**, choose **Simple Animation**.
4. Under **Simple Animation**, click the + button to add an element. Select an animation to play.
5. Open the **Particle Editor** and drag the effect that you want to preview into your level.
6. In the **Entity Inspector**, click **Add Component**. Under **Animation**, choose **Attachment**.
7. Under **Attachment**, click the target entity picker and select your character in the viewport.
8. In the joint list, select a joint name to which to attach the effect. Select an offset as needed.
9. In the bottom toolbar, click **AI/Physics**. Alternatively, press **Ctrl+G** to start the animation and preview the effect.

### Generating Particles from Surface Properties

Use the properties for an object's material surface to define the event-driven effects that can occur when an object experiences an event. You can specify these events on a render material or on individual pieces or surfaces of a `.cgf` asset.

Many of the properties define the particle effects that are spawned by events such as walking or a bullet hit. To specify the effect that is spawned when a geometry piece breaks off of an object, set the following parameters in a Lua script:

**Name**
- Specifies the name of the particle effect.

**Scale**
- Multiplies the size of the particle.

**Count_scale**
- Multiplies the particle counts.

### Particle Debugging with Console Variables

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following console variables to monitor and debug particle system issues. For more information, see Using the Console Window (p. 210).

**Particle Console Variables**

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>e_Particles</code></td>
<td>Activates drawing of particles.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td><code>e_ParticlesAllowRuntimeLoad</code></td>
<td>Allows loading of dynamic particle effects at runtime.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td><code>e_ParticlesAnimBlend</code></td>
<td>Blends between animated texture frames.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Force</td>
</tr>
<tr>
<td><code>e_ParticlesAudio</code></td>
<td>Toggles audio for particles.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td><code>e_ParticlesCullAgainstOcclusionBuffer</code></td>
<td>particles against the occlusion buffer.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
</tbody>
</table>
### Particle Debugging with Console Variables

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_ParticlesCullAgainstViewFrustum</td>
<td>Culls particles against the view frustum.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>e_ParticlesDebug</td>
<td>Displays the particle counts on the screen.</td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td></td>
<td>You can also specify the particle debug flags that you want to add or remove.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To add or remove a particle debug flag, you can use the Console command line or the .cfg files. Type the console variable followed by the flag and a + or - .</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For example, if you want to use the m flag to show memory usage, enter the following command: e_ParticlesDebug b+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To remove the m flag, enter the following command: e_ParticlesDebug b-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information, see Using the Console Window (p. 210).</td>
<td></td>
</tr>
<tr>
<td>e_ParticleDumpMemorAfterMapLoad</td>
<td>Specify a value of 1 to dump particle memory after map load.</td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td>e_ParticlesGI</td>
<td>Applies global illumination to appropriate particle effects.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Force</td>
</tr>
<tr>
<td>e_ParticleShadowsNumGSMs</td>
<td>Number of global shadow maps (GSM) used for particle shadows.</td>
<td>Default value: 3</td>
</tr>
<tr>
<td>e_ParticlesIndexPoolSize</td>
<td>Memory size of index pool between particle and render thread.</td>
<td>Default value: 16</td>
</tr>
<tr>
<td>Console Variable</td>
<td>Description</td>
<td>Valid Values</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>e_ParticlesLightMinColorThreshold</td>
<td>Threshold for minimum particle light color.</td>
<td>Default value: 0</td>
</tr>
<tr>
<td>e_ParticlesLightMinRadiusThreshold</td>
<td>Threshold of minimum particle light radius.</td>
<td>Default value: 0</td>
</tr>
<tr>
<td>e_ParticlesLights</td>
<td>Allows a light source to be attached to every particle.</td>
<td>0 = Off 1 = Deferred lights</td>
</tr>
<tr>
<td>e_ParticlesLightsViewDistRatio</td>
<td>Sets the view distance ratio for particle lights.</td>
<td>Default value: 256</td>
</tr>
<tr>
<td>e_ParticlesLod</td>
<td>Multiplier to particle count.</td>
<td>Default value: 0</td>
</tr>
<tr>
<td>e_ParticlesMaxDrawScreen</td>
<td>Screen size maximum per particle. Particles that reach this limit fade out, even if the particle does not reach its lifetime.</td>
<td>Default value: 256</td>
</tr>
<tr>
<td>e_ParticlesMaxScreenFill</td>
<td>Screen size maximum of total particles to draw.</td>
<td>Default value: 160</td>
</tr>
<tr>
<td>e_ParticlesMinDrawAlpha</td>
<td>Alpha cutoff for rendering particles.</td>
<td>Default value: 0.004</td>
</tr>
<tr>
<td>e_ParticlesMinDrawPixels</td>
<td>Pixel size minimum per particle. Particles that reach this limit fade out, even if the particle does not reach its lifetime.</td>
<td>Default value: 1</td>
</tr>
<tr>
<td>e_ParticlesMotionBlur</td>
<td>Enables motion blur for particles.</td>
<td>0 = Off 1 = On (default) 2 = Force</td>
</tr>
<tr>
<td>e_ParticlesObjectCollisions</td>
<td>Enables particle and object collisions for the SimpleCollision parameter. For more information, see Collision Attribute (p. 1517).</td>
<td>1 = Against static objects only. 2 = Against dynamic objects also. (default)</td>
</tr>
<tr>
<td>e_ParticlesPoolSize</td>
<td>Particle system pool memory size in KB.</td>
<td>Default value: 16384</td>
</tr>
<tr>
<td>e_ParticlesPreload</td>
<td>Enables preloading of all particle effects at the beginning.</td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>
### Console Variable Description

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
</table>
| e_ParticlesProfile | Displays particle information on the screen. | 0 = Off (default)  
1 = Always show the statistics about particle pool usage.  
2 = Disable the warning message when running out of pool memory. |
| e_ParticlesQuality | Specifies the quality of particle detail. | Default value: 4 |
| e_ParticlesShadows | Displays shadows on particles. | 0 = Off  
1 = On (default)  
2 = Force |
| e_ParticlesShowMainThreadUpdates | Renders a list of containers not updated by a job and provides details why. | Default value: 1 |
| e_ParticlesSoftIntersect | Renders appropriate particles with soft intersection. | 0 = Off  
1 = On (default)  
2 = Force |
| e_ParticlesSortQuality | Minimum sort quality for new particle insertion. | 0 = Basic (default)  
1 = Better  
2 = Best |
| e_ParticlesThread | Enables particle threading. | 0 = Off  
1 = On (default) |
| e_ParticlesUseLevelSpecificLibs | Allows searching for level-specific version of effects files. | 0 = Off (default)  
1 = On |
| e_ParticlesVertexPoolSize | Memory size of vertex pool between particle and render thread. | Default value: 256 |
| e_SkipParticleOcclusion | Skips occlusion testing for particles in the occlusion buffer. **Note** The particles may be too small to return reliable results. | 0 = Off  
1 = On (default) |
| gpu_particle_physics | Enables graphics processing unit (GPU) for physics, if available. | 0 = Off (default)  
1 = On |
<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_breakage_particles_limit</td>
<td>Imposes a limit on particles generated during 2D surfaces breaking.</td>
<td>Default value: 160</td>
</tr>
<tr>
<td>mfx_ParticleImpactThresh</td>
<td>Impact thread for particle effects.</td>
<td>Default value: 2</td>
</tr>
<tr>
<td>r_GPU ParticleDepthCubemapResolution</td>
<td>Resolution for the cubemaps used by the cubemap depth collision feature for GPU particles.</td>
<td>Default value: 256</td>
</tr>
<tr>
<td>r_ParticlesAmountGI</td>
<td>Global illumination (GI) amount for particles without an assigned material.</td>
<td>Default value: 0.15</td>
</tr>
<tr>
<td>r_ParticlesDebug</td>
<td>(Advanced) Evaluates what particles on screen may impact performance.</td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Display particle screen coverage.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Display particle overdraw.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r_ParticlesHalfRes</td>
<td>Enables or forces the render of particles in a half-resolution buffer.</td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Force</td>
</tr>
<tr>
<td>r_ParticlesHalfResAmount</td>
<td>Specifies the particle half-resolution buffer to a half or quarter of the screen size.</td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Half of screen size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Quarter of screen size</td>
</tr>
<tr>
<td>r_ParticlesHalfResBlendMode</td>
<td>Specifies which particles can be rendered in half resolution.</td>
<td>0 = Alpha (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Additive</td>
</tr>
<tr>
<td>r_ParticlesInstanceVertices</td>
<td>Enables instanced-vertex rendering.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>r_ParticlesRefraction</td>
<td>Enables refractive particles.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>Console Variable</td>
<td>Description</td>
<td>Valid Values</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>r_ParticlesSoftIsec</td>
<td>Enables particle soft intersections.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>r_ParticlesTessellation</td>
<td>Enables particle tessellation for higher quality lighting (DX11 only).</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>r_ParticlesTessellationTriSize</td>
<td>Sets the particle tessellation triangle screen space size in pixels (DX11 only).</td>
<td>Default value: 16</td>
</tr>
<tr>
<td>r_ParticlesVerticePoolSize</td>
<td>Maximum number of particle vertices to support.</td>
<td>Default value: 15360</td>
</tr>
<tr>
<td>r_ShadowParticlesAnimJitterAmount</td>
<td>Amount of animated jittering for particle shadows.</td>
<td>Default value: 1</td>
</tr>
<tr>
<td>r_ShadowsParticlesJitterAmount</td>
<td>Amount of jittering for particle shadows.</td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>r_ShadowsParticlesKernelSize</td>
<td>Blurs kernel size for particle shadows.</td>
<td>0 = Hard edge</td>
</tr>
<tr>
<td></td>
<td>Specifying higher values for more blur.</td>
<td>Default value: 1</td>
</tr>
<tr>
<td>r_ShadowsParticleNormalEffect</td>
<td>Shadow taps on particles affected by normal and intensity (breaks lines and uniformity of shadows).</td>
<td>Default value: 1</td>
</tr>
<tr>
<td>sys_spec_particles</td>
<td>The console variable group to apply settings for multiple console variables.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Specify a number to switch the group of particle console variables. You can find the group number, the specified console variables, and their values in the sys_spec_Particles.cfg file in the lumberyard_version\dev\Engine\Config\CVarGroups directory.

For example, when sys_spec_particles = 1, the console variable uses the settings defined in group 1 in the sys_spec_Particles.cfg file.

---

Create cinematic sequences

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cinematics, also known as sequences or cutscenes, are interactive movie animations with time-dependent control over objects and events. You can use Lumberyard to add cutscenes to your game.
You can also add scripted events so that a sequence of objects, animations, and sounds are triggered in the game. The player can view these sequences from their own (first person) or another’s (third person) perspective.

Sequences consist of the following elements (listed in hierarchical order), which are created and managed from the Track View:

- **Node** – Each sequence comprises a top-level director (scene) node, one or more camera nodes, image effects nodes, and entity nodes.
- **Track** – Depending on the type, each node consists of multiple tracks, such as position, animation, sound, lighting, text, and events. Tracks are displayed in the track timeline pane.
- **Key** – A key is a setting for a property at a specific time. As the sequence plays, keys are interpolated based on their in and out tangent values set in the Track View.

**Topics**
- Using the Track View Editor (p. 1584)
- Populating a Scene (p. 1608)
- Animating Characters in Scenes (p. 1622)
- Capturing Image Frames (p. 1630)
- Debugging Cinematic Scenes with Console Variables (p. 1632)

**Using the Track View Editor**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Track View is the primary tool to create and manage cinematic sequences. A *sequence* is the content generated from the Track View for cutscenes or other canned animation triggers. When you create a sequence, this creates a component entity in the level. The component entity stores all of the animation key data that you specified in the Track View.

If you want to generate cutscenes for games or create a script to trigger an animation, you can use the Track View to control cameras, component entities, global variables in a level, and so on.

**To create a sequence in the Track View**

1. Do one of the following:
   - In Lumberyard Editor, choose Tools, Track View.
   - Press T.
2. To create a sequence, do one of the following:
   - Choose Sequences, New Sequence.
3. Enter a sequence name, such as Example Sequence and click OK.
4. In the Entity Outliner, a component entity appears with the same name as your sequence. This component entity has a Sequence component that stores your sequence data from the Track View.
After you create a sequence, you can add properties to it. Any part of the sequence is considered a node. Nodes can be a reference to existing component entities or added to a sequence. For example, if you want to include an active camera for your sequence, you can add the Director node. Each node can have one or more tracks, depending on your animation sequence. A track displays animation keys on a timeline in relation to the property that is animated on a node.

To add a node to a sequence

1. In the Track View, right-click the sequence or the node browser and select Add Name Node.
2. Select the node to update its properties.

For more information, see Track View Editor Nodes (p. 1586).

Topics

- Track View Editor Toolbars (p. 1585)
- Track View Editor Nodes (p. 1586)
- Adding and Removing Animation Keys on Tracks (p. 1601)
- Controlling the Playhead (p. 1602)
- Using Record Mode (p. 1604)
- Using Animation Curves (p. 1607)

Track View Editor Toolbars

See the following main sections of the Track View:

1. Main toolbar – Tools to create and modify sequences and the tracks in a sequence. The main toolbar includes the Sequence/Node, View, Play, Keys, and Tracks toolbars. To add or remove toolbars, right-click the main toolbar and select your preferred options.
2. Node browser – Tree pane of all nodes and associated tracks. Nodes can be in reference to existing component entities or nodes that you can add to a sequence such as a Director Node.
3. Track Editor – Track timeline of all node tracks specified for animation keys. Each row in the timeline corresponds to a track listed in the node browser.
4. Key properties – Pane that shows more information for a selected animation key. It also provides information about the key number on the track and its location in the timeline, in seconds. Common properties are color (RGB) values and floats.
5. Curve Editor – Pane for controlling keys and their interpolation for all sequence nodes. The Curve Editor provides more control over easing in or out of curves, setting some curves to be linear, and so on, depending on your animation requirements.

For more information, see Using Animation Curves (p. 1607).

Note

You can pause over buttons in the Track View to view their descriptions.
Track View Editor Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

The Track View offers a variety of nodes for specific purposes. Anything part of the sequence is considered a node. Nodes can be a reference to existing component entities or added to a sequence. The top-level node in the tree view is the sequence and all other nodes are listed below the sequence.

Topics

- Component Entities and Component Nodes (p. 1586)
- Comment Node (p. 1590)
- Console Variable Node (p. 1590)
- Director (Scene) Node (p. 1591)
- Environment Node (p. 1594)
- Event Node (p. 1595)
- Material Node (p. 1596)
- Shadows Setup Node (p. 1598)
- Full Screen Effect Nodes (p. 1598)

Component Entities and Component Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.
Topics

- Naming and Identifying Component Entities (p. 1588)
- Adding or Removing Components from a Component Entity (p. 1589)
- Animating Components on a Component Entity (p. 1589)

In the Track View, component entity nodes function as containers for component nodes.

When you add an animation using the Track View, the animation track is applied to a component node. Component entity nodes don't directly have tracks or key properties.

Example

The component entity GameObject contains Transform, Mesh, and Point Light components. When you add the GameObject component entity to a sequence in the Track View, you can see all the components in the node browser. The component entity node is a reference to which components are animated in the sequence.

Component nodes that can be animated are nested as children under the associated entity node. You can add animation tracks to any of these component nodes.

To add a track to an animation node

1. In the Track View, create or select a sequence.
2. In the node browser, right-click and choose Add Track, then choose the component property.
Not all components can be animated in the Track View. For example, you can add the Visibility track only for the Mesh component. The Point Light component has multiple tracks that you can add to the sequence. In the following example, the Color, DiffuseMultiplier, and Visible tracks are added to the sequence.

For more information about adding node support to the Track View, see Exposing Custom Components to Track View for Animation (p. 1021).

**Naming and Identifying Component Entities**

Lumberyard uses entity IDs to identify component entities, which means that you can name your component entities as you want. This includes reusing the same name for multiple entities. In the Track View, if component entity nodes share the same name, a number is appended to the name. This doesn't change the name of the component entity in the level, but it might be difficult to determine which entity to animate.
Example

If you have component entities with the same name (for example, if they're multiple instances of a slice), you can determine which entity that you're editing. For more information, see Working with Slices and Sequences (p. 1611).

Adding or Removing Components from a Component Entity

When you add a component to a component entity in Lumberyard Editor, the component is automatically added to any component entity nodes in the Track View. When you remove a component, the component and any animation data are also removed from the Track View.

**Important**

Be careful when removing components from component entities because it might affect your existing sequence. For example, if you remove a **Simple Motion** component from an entity that is part of a sequence, the animation no longer references the specified animation.

Animating Components on a Component Entity

Component nodes that can be animated are nested as children under the associated component entity node. You can add animation tracks to any of these component nodes.

**To add an animation track to a component node**

1. In the Track View, select or create a sequence.
2. In the node browser, right-click the component node and choose **Add Track**, then choose the track that you want.
Using the Track View Editor

Note

- Some components support only a limited number of tracks that can be animated in a sequence. For more information for component-specific properties, see the Component Reference (p. 532).
- Not all components can be animated in the Track View. For more information about adding node support to the Track View, see Exposing Custom Components to Track View for Animation (p. 1021).

After you add the component entity nodes to the sequence and specify the tracks to animate, you can then add keyframes to the timeline. In keyframes, you specify where in the timeline you want to animate the property and edit its properties.

For more information, see Adding and Removing Animation Keys on Tracks (p. 1601).

Comment Node

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Comment node to add comments to your track view sequence. This is mostly used for production purposes and is not rendered in the game.

To add a Comment node in the Track View

1. In the Track View, right-click either the sequence (top node) or the Director node in the tree as applicable, and then click Add Comment Node.
2. For each of the keys listed below, click the applicable key listed under the Comment node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under Key Properties enter a value for Value.

Comment Node Key Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Pos X</td>
<td>position of the text horizontally</td>
</tr>
<tr>
<td>Unit Pos Y</td>
<td>position of the text vertically</td>
</tr>
</tbody>
</table>
| Text | • Comment – Text string  
  • Duration – Length of time the node is active  
  • Size – Font size  
  • Color – Font color  
  • Align – Text alignment (Center, Left, Right)  
  • Font – Font type (default, console, hud) |

Console Variable Node

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Use the **Console Variable** node to use and animate console variables in a track view sequence.

**To add a console variable node in Track View**

1. In the Track View, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Console Variable**. Type a name for it and click **OK**.
2. At the bottom of **Lumberyard Editor**, right-click the text box in the **Console** window, which opens up the **Console Variables** window that displays a list of all available console variables.
3. Pause on the desired console variable to get a tool tip that gives a description and valid values to use.
4. In the Track View select the **value** key listed under the console variable node.
5. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties** enter a value for **Value**.

**To animate a console variable**

1. In the Track View click **View, Curve Editor**.
2. Click **Set In Tangent To Step** button (located third button from the left above the timeline window) to set the keyframes for the console variable.

**Director (Scene) Node**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Director (Scene)** node includes a camera track that specifies the active camera for a track view sequence. You can add sequence-specific nodes (for example, **Depth of Field** or **Comment**) under the **Director** node to override the same nodes that were set at the sequence level.

**To add a Director node in the Track View**

1. In Lumberyard Editor, choose **Tools, Track View**.
2. In the Track View, click the **Add Sequence** icon.
3. In the **Add New Sequence** dialog box, enter a name for your sequence and click **OK**.
4. Right-click your sequence and choose **Add Director (Scene) Node**.
5. Right-click the **Director** node and click **Add Track**.
6. Select the track and double-click to position the key on its highlighted row in the timeline.
7. Double-click the green marker, and under **Key Properties**, enter a value for **Value**.

You can add the following tracks and then set the key properties to the **Director** node.

### Director Node Tracks and Key Properties

<table>
<thead>
<tr>
<th>Track</th>
<th>Key Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>Camera</td>
<td>Specifies the sequence camera. You can optionally override the camera with the <code>mov_overrideCam</code> console variable. Set the console variable to an entity ID (string) of a <strong>Camera</strong> (p. 566) component entity for component entity sequences. For more information, see Using the Console Window (p. 210).</td>
</tr>
<tr>
<td></td>
<td>Blend time</td>
<td>Specifies the blend time, in seconds, between sequential cameras in the track.</td>
</tr>
<tr>
<td>Capture</td>
<td>Duration</td>
<td>Sets the capture duration in seconds.</td>
</tr>
<tr>
<td></td>
<td>Time Step</td>
<td>Forces a fixed frame rate, in seconds, by using a specified time step. Time step = 1/number of frames. Units are in 1 fps (frames per second), so a time step value of .0333 results in a game frame rate of 30 fps.</td>
</tr>
<tr>
<td></td>
<td>Output Format</td>
<td>Specifies the image output file format. Choose from the following options:</td>
</tr>
</tbody>
</table>
## Track View Editor

<table>
<thead>
<tr>
<th>Track</th>
<th>Key Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Output Prefix</strong></td>
<td>Specifies a prefix to apply to the image file names. For best results, use the same prefix as the sequence for clarity.</td>
</tr>
<tr>
<td></td>
<td><strong>Output Folder</strong></td>
<td>Specifies the directory where the image files are stored.</td>
</tr>
<tr>
<td></td>
<td><strong>Buffer(s) to capture</strong></td>
<td>Specifies the image capture format. Choose from the following options:</td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>RGB pixel information</td>
</tr>
<tr>
<td></td>
<td>Color+Alpha</td>
<td>RGB pixel data and Alpha channel data</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
<td>To capture Alpha channel data, you must set <strong>Output Format to tga.</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Just one frame?</strong></td>
<td>Specifies if you want to capture a single frame.</td>
</tr>
<tr>
<td><strong>Console</strong></td>
<td><strong>Command</strong></td>
<td>Runs the specified console variable command.</td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td><strong>Event</strong></td>
<td>Specifies the event to trigger in the <strong>Director</strong> node Lua script.</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
<td>Specifies the custom value that is referenced in the nodes for your Script Canvas graph or code.</td>
</tr>
<tr>
<td></td>
<td><strong>No trigger in scrubbing.</strong></td>
<td>Prevents the event from occurring while you preview the timeline of the track view sequence.</td>
</tr>
<tr>
<td><strong>FixedTimeStep</strong></td>
<td><strong>Value</strong></td>
<td>Sets a fixed time step in order to modify the game speed. Units are in 1 fps, so a fixed time step value of .0333 results in a game frame rate of 30 fps.</td>
</tr>
<tr>
<td><strong>GoTo</strong></td>
<td><strong>Goto Time</strong></td>
<td>Jumps forward or backward in a sequence. Use this track for key framing time shifts and to loop part of a sequence. This key automatically applies animation blending on animations that are currently playing in the sequence.</td>
</tr>
<tr>
<td><strong>Sequence</strong></td>
<td><strong>Sequence</strong></td>
<td>Specifies the track view sequence to play at the specified keyframe.</td>
</tr>
<tr>
<td></td>
<td><strong>Override Start/End Times</strong></td>
<td>Overrides the sequence start and end times.</td>
</tr>
<tr>
<td></td>
<td><strong>Start Time</strong></td>
<td>Specifies the start time to override.</td>
</tr>
<tr>
<td></td>
<td><strong>End Time</strong></td>
<td>Specifies the end time to override.</td>
</tr>
<tr>
<td><strong>Sound</strong></td>
<td><strong>StartTrigger</strong></td>
<td>Specifies the trigger to start the audio.</td>
</tr>
</tbody>
</table>
Using the Track View Editor

### Track Key Property

<table>
<thead>
<tr>
<th>Track</th>
<th>Key Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>StopTrigger</td>
<td>Specifies the trigger to stop the audio.</td>
</tr>
<tr>
<td></td>
<td>Duration</td>
<td>Specifies the length to play the audio.</td>
</tr>
<tr>
<td></td>
<td>Custom Color</td>
<td>Specifies the color to display for the audio on the timeline.</td>
</tr>
</tbody>
</table>

**Timewarp**

|       | Value | Creates a slow-motion effect using a curve that scales the movie playback speed by the Timewarp value. A value of 1.0 represents normal speed; smaller values slow down time and larger values speed up time. If you set Timewarp to 0, playback will completely stop in your movie. Timewarp applies only to visuals; sounds are not slowed down. |

**Note**

- You can add multiple Director nodes to a scene, but only one Director node can be active.
- To change the active Director node, right-click the node and choose Set as Active Director. When you deactivate a Director node, all child node animations are deactivated. This is useful when you want to enable and disable animation for specific objects within the same track view sequence.

### Environment Node

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Environment node to set the sun's longitude and latitude in a scene.

**To add an Environment node in the Track View**

1. In the Track View, right-click either the sequence (top node) or the Director node in the tree as applicable, and then click Add Environment Node.
2. For each of the keys listed below, click the applicable key listed under the Environment node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under Key Properties, enter a value for Value.

### Environment Node Key Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Longitude</td>
<td>Sets the sun's longitude.</td>
</tr>
<tr>
<td>Sun Latitude</td>
<td>Sets the sun's latitude.</td>
</tr>
</tbody>
</table>
Event Node

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add an Event node to your sequence to trigger and send values to Script Canvas. You create Track Events using the Track View Events window. You then assign the Track Event to an animation key that is added to the track for the Event node. When the key is played during the sequence, the event is triggered. Script Canvas then uses the Track Event to trigger additional script logic.

To add an Event node

1. In the Track View, select or create a sequence.
2. In the node browser, right-click and choose Add Event Node.
3. In the Track Event Name window, enter a name for the track and choose OK.

This creates a Track Event node and a Track Event track is automatically to that node.

To create a Track Event

1. In the node browser, right-click and choose Edit Events. This opens the Track View Events window.
2. In the **Track View Events** window, click **Add** to create an event.

3. Enter an event name and choose **OK**. Your track event appears in the window.

4. When you are done, close the window. You can now specify this track event in Script Canvas.

**Material Node**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Material** nodes help you animate commonly used material properties that you would normally set in the **Material Editor**. You can add **Material** nodes through a sequence or from the **Director** node.

**Note**

The name of the **Material** node must be the full path of the material that you want to animate, as shown in the **Material Editor**.
A recommended workflow is to select the material inside the **Material Editor** that you want to animate.

**To add a Material node in the Track View**

1. In Lumberyard Editor, choose **Tools, Material Editor**.
2. Navigate to the material that you want to animate.
3. Right-click the selected material and choose **Copy Path to Clipboard**. If the material is in a material group, select the group and copy the group name to the clipboard.

![Material Editor](image)

4. Choose **Tools, Track View**.
5. Create or select an existing sequence or **Director** node that you want to contain the animation.
6. In the node tree, right-click and choose **Add Material Node** and then in the **Material Name** dialog box, press Ctrl+V to paste the full path to the material that you copied in step 3 and then click **OK**.
7. If the material is in a material group, the text appears red on the **Material** node in the Track View. This means that a submaterial hasn't been selected.

To add a submaterial, do the following:

a. In the Track View, right-click the **Material** node.
b. Select the submaterial that you want to animate.
c. The **Material** node text should no longer appear in red.

8. To add tracks to the **Material** node, right-click the node name and choose **Add Track**. See the following table for the available tracks.

### Material Node Tracks

<table>
<thead>
<tr>
<th>Track</th>
<th>Key Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffuse</td>
<td>Color (RGB)</td>
<td>RGB values to specify the base color of a material.</td>
</tr>
<tr>
<td>Emissive Color</td>
<td>Color (RGB)</td>
<td>RGB values to enable objects to emit light and be visible in the dark.</td>
</tr>
<tr>
<td>Emissive Intensity</td>
<td>Float (0.00 to 1.0)</td>
<td>Float value that controls the brightness simulating light emitting from the surface that makes an object glow.</td>
</tr>
<tr>
<td>Emittance Map Gamma</td>
<td>Float (1.0 to 2.0)</td>
<td>Float value that expands the lower range of the emittance map. This makes darker colors appear less bright.</td>
</tr>
<tr>
<td>Glossiness</td>
<td>Float (0 to 255)</td>
<td>Acuity or sharpness of the specular reflection. Values of 10 or less create a scattered reflection. Values greater than 10 create a sharp reflection.</td>
</tr>
</tbody>
</table>
## Using the Track View Editor

<table>
<thead>
<tr>
<th>Track</th>
<th>Key Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect Color</td>
<td>Color (RBG)</td>
<td>RGB values to specify that tints the bounce light from global illumination.</td>
</tr>
<tr>
<td>Opacity</td>
<td>Float (0.00 to 1.0)</td>
<td>Degree of transparency. Values less than 50 fall more to the white end of the alpha channel map. Values greater than 50 fall more to the black end of the alpha channel map.</td>
</tr>
</tbody>
</table>
| SSSIndex            | Float (0.00 to 3.99) | Controls subsurface scattering profile and amount.  
For marble, specify a value between 0.01 to 0.99.  
For skin, specify a value between 1.00 to 1.99. |
| Specular            | Color (RGB)  | Reflective brightness and color of a material when light shines on the object. The greater the value, the shinier the material.  
To apply reflections in degrees of black and white, specify the same values for R, G, and B. For colored reflections, specify different values for each. |

### Shadows Setup Node

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.  
Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Shadows Setup node to add or remove sun shadow maps over several frames in a sequence.

**To add a Shadows Setup node in the Track View**

1. In the Track View, right-click the sequence (top node) or the Director node in the tree, and then choose Add Shadows Setup Node.
2. Select the GSMCache key under the ShadowsSetup node.
3. Double-click to position the key on its highlighted row in the timeline.

### Full Screen Effect Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.  
Download O3DE or visit the AWS Game Tech blog to learn more.

Full Screen Effect nodes create post-processing effects for a sequence. They are added by using the context menu for a Sequence or Director node.

**Topics**

- Radial Blur Node (p. 1599)
You can use the Radial Blur node to blur the animation radially outward from a center point.

**To add a Radial Blur node in the Track View**

1. In the Track View, right-click the sequence (top node) or the Director node in the tree as applicable, and then choose Add Radial Blur Node.
2. For each of the keys in the following list, click the applicable key listed under the RadialBlur node.
3. Double-click to position the key on its highlighted row in the timeline.
4. Double-click the green marker, and under Key Properties, enter a value for Value.

**Radial Blur Node Key Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amount</strong></td>
<td>Intensity of the blur effect. Range is 0 to 1.</td>
</tr>
<tr>
<td><strong>ScreenPosX</strong></td>
<td>X-axis position of the effect's center. The range is −1 to 1, with 0.5 being the center of the screen.</td>
</tr>
<tr>
<td><strong>ScreenPosY</strong></td>
<td>Y-axis position of the effect's center. The range is −1 to 1, with 0.5 being the center of the screen.</td>
</tr>
<tr>
<td><strong>Blurring Radius</strong></td>
<td>Size of the blur effect. The range is 0 (not visible) to 1 (covers the entire screen).</td>
</tr>
</tbody>
</table>

**Color Correction Node**

You use the Color Correction node to change the CMYK, brightness, contrast, saturation, and hue in a scene. Most color correction properties don't update smoothly. For this reason, you should hide stronger color correction changes should by cuts or fading between scenes.

**To add a Color Correction node in the Track View**

1. In the Track View, right-click either the sequence (top node) or the Director node in the tree as applicable, and then click Add Color Correction Node.
2. Click the applicable key listed under the ColorCorrection node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under Key Properties enter an applicable value for Value.
**Note**
To have change correction dynamically based on a variable, you can use the **Enable Color Correction** node in the **Script Canvas** editor.

### Adding a Depth of Field Node

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Depth of Field** (DOF) node to add realism to scenes to simulate the way a real-world camera works. You can use a broad depth of field to focus on the entire scene, or use a shallow depth of field to have sharp focus only on objects that are a specific distance from the camera.

**To add a Depth of Field node in the Track View**

1. In the Track View, right-click the sequence (top node) or the **Director** node in the tree as applicable, and then choose **Add Depth of Field Node**.
2. For each of the keys listed below, click the applicable key listed under the **DepthOfField** node.
3. Double-click to position the key on its highlighted row in the timeline.
4. Double-click the green marker, and under **Key Properties**, enter a value for **Value**.

**Depth Of Field Node Key Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Enables or disables depth of field effect.</td>
</tr>
<tr>
<td>FocusDistance</td>
<td>Distance the focus is from the camera. Positive values are in front of the camera while negative values are behind the camera.</td>
</tr>
<tr>
<td>FocusRange</td>
<td>Distance toward and away from the camera until maximum blurriness is reached. By default, this value is twice the <strong>FocusDistance</strong> value.</td>
</tr>
<tr>
<td>BlurAmount</td>
<td>Maximum blurriness value.</td>
</tr>
</tbody>
</table>

**Screen Fader Node**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the **Screen Fader** node to fade the screen in and out in a scene.

**To add a Screen Fader node in the Track View**

1. In the Track View, right-click the sequence (top node) or the **Director** node in the tree as applicable, and then choose **Add Screen Fader**.
2. Click the **Fader** key under the **ScreenFader** node.
3. Double-click to position the key on its highlighted row in the timeline.
4. Double-click the green marker, and under **Key Properties**, enter a value for **Value**.
Screen Fader Node Key Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Specifies either FadeIn or FadeOut values.</td>
</tr>
<tr>
<td>ChangeType</td>
<td>For this transition type, you can specify one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Cubic Square</td>
</tr>
<tr>
<td></td>
<td>• Linear</td>
</tr>
<tr>
<td></td>
<td>• Sinus Curve</td>
</tr>
<tr>
<td></td>
<td>• Square</td>
</tr>
<tr>
<td></td>
<td>• Square Root</td>
</tr>
<tr>
<td>Color</td>
<td>Specifies the RGB value used for fading.</td>
</tr>
<tr>
<td>Duration</td>
<td>Specifies how long it takes to fade in or out the screen.</td>
</tr>
<tr>
<td>Texture</td>
<td>Specifies a texture file to use as a screen overlay. An alpha texture is</td>
</tr>
<tr>
<td></td>
<td>commonly used for effects like dirt or blood. The texture is multiplied by</td>
</tr>
<tr>
<td></td>
<td>the color value to allow you to animate the brightness during the fade.</td>
</tr>
<tr>
<td>Use Current Color</td>
<td>Ignores the Color property and uses the color of the previous key instead.</td>
</tr>
</tbody>
</table>

Adding and Removing Animation Keys on Tracks

You can add a keyframe to a track in the timeline with one of the following:

**Double-click the track**

This is the quickest way to add an animation key to the track. This adds a key exactly where you click and stores the data at the exact point in the timeline.

**Add Keys**

In the Keys toolbar, click the Add Keys icon and then click the timeline. This is useful if you need to add many keys to the timeline. To stop adding keys, choose the move, scale, or slide icon.

**Record Mode**

In the Play toolbar, click the Record Mode icon and then makes changes directly to your components.

When you enter record mode, you can update the component entities that are part of the sequence in the viewport. Animation keys are added automatically to the appropriate node tracks in the timeline, based on the current location of the timeline playhead.

For example, if you specify a different value for the Transform component at three seconds, the key for this update appears in the timeline.

To specify an animation to play over time, move the playhead to different places along the timeline. Otherwise, you will overwrite the keys at the same location of the timeline as you update the component entities in the level.
To stop recording, click the **Record Mode** icon again.

For more information, see Using Record Mode (p. 1604).

You can delete individual keys or click and drag to select multiple animation keys.

**To delete keys from your timeline**

1. In the timeline, select a key.
2. Do one of the following:
   - Right-click the key and choose **Delete**.
   - Press **Delete** on your keyboard.
   - In the **Keys** toolbar, click the **Delete Keys** icon.

---

### Controlling the Playhead

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Track View's **Play** toolbar contains the main controls for controlling the playhead for the sequence timeline.

<table>
<thead>
<tr>
<th>Toolbar Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to start of sequence</td>
<td>Playhead is moved to the <strong>In</strong> marker of your sequence. For more information, see Setting the In and Out Markers (p. 1603).</td>
</tr>
<tr>
<td>Play</td>
<td>Activates play mode and moves the playhead forward through the sequence timeline. Click the drop-down menu to control the play speed.</td>
</tr>
<tr>
<td></td>
<td>The default is <strong>1x</strong> speed. Other options include <strong>2x</strong>, <strong>1/2x</strong>, <strong>1/4x</strong>, and <strong>1/8x</strong>.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops the playhead. Click the drop-down menu to specify the default <strong>Stop</strong> or <strong>Stop with Hard Reset</strong>, which returns the playhead to the start of the sequence.</td>
</tr>
<tr>
<td>Pause</td>
<td>During play mode, holds the playhead at the current point in the timeline. When selected again, the sequence resumes playing.</td>
</tr>
<tr>
<td>Go to end of sequence</td>
<td>Playhead is moved to the <strong>Out</strong> marker of the sequence.</td>
</tr>
</tbody>
</table>
Using the Track View Editor

### Toolbar Options

<table>
<thead>
<tr>
<th>Description</th>
<th>For more information, see Setting the In and Out Markers (p. 1603).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start Animation Recording</strong></td>
<td>Also known as record mode, you can manipulate objects within the level and the animation keys are automatically added to the sequence. For more information, see Using Record Mode (p. 1604).</td>
</tr>
<tr>
<td><strong>Start Auto Recording</strong></td>
<td>When selected, also activates record mode. The playhead automatically moves in the timeline based on the setting specified in the drop-down menu. The default is 1 sec. Other options include 1/2 sec, 1/5 sec, 1/10 sec, 1/25 sec, 1/50 sec, and 1/100 sec.</td>
</tr>
<tr>
<td><strong>Loop</strong></td>
<td>During play mode, when the playhead reaches the End marker it returns to the In marker of the sequence and continues playing the sequence again.</td>
</tr>
<tr>
<td><strong>Playhead Location</strong></td>
<td>Shows information about the playhead's current location in the timeline and the assigned frame rate.</td>
</tr>
<tr>
<td><strong>Frame Rate</strong></td>
<td>Assigns the frame rate for the sequence. This number is also used for frame snapping.</td>
</tr>
<tr>
<td><strong>Active Camera/Camera Name</strong></td>
<td>Camera name is updated to show which camera is active from the Director Node only if the Editor Viewport Camera is set to Sequence Camera.</td>
</tr>
<tr>
<td><strong>Undo</strong></td>
<td>Reverts the previous action.</td>
</tr>
<tr>
<td><strong>Redo</strong></td>
<td>Applies the previous action.</td>
</tr>
</tbody>
</table>

**Note**
You can also manually adjust the playhead along the timeline. At the top of the sequence timeline, select and drag the playhead to your preferred location.

### Setting the In and Out Markers

The In and Out markers are small red triangles at the top of the sequence timeline. By default, the In and Out markers are set to the beginning and end of the sequence. You can move the markers by right-clicking at the top of the timeline where the frames/seconds are listed. Depending on where you right-click, the Track View identifies which marker is closest to the mouse cursor and moves that one.

For example, if a sequence is 300 frames long, and you right-click at the 50 frame tick, the In marker will move to that spot. If you right-click at the 200 frame tick, the Out marker will move to that spot.
Note
When you move the In and Out markers, this will apply the updated range to the Go to start of sequence, Play, Go to end of Sequence, and Loop settings. For example, if you change the In marker to the two second mark, the Go to start of sequence icon will now move the playhead to the two second mark.

Using Record Mode

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Track View's Play toolbar has the Start Animation Recording icon to enter record mode. In record mode, you can manipulate component entities for a sequence directly in the viewport. Animation keys are automatically placed on the appropriate node tracks in the timeline based on the current location of the timeline playhead.

To use record mode

1. In the Track View, select or create the sequence that you want.
2. In the Play toolbar, click the Start Animation Recording icon.
3. In the viewport, manipulate a component entity that is part of your sequence. Your changes will appear in the timeline as animation keys.
4. To stop recording, click the Start Animation Recording icon again.

Note
You must move the playhead to different locations on the timeline in order to have an animation play over time. Otherwise, you will override the keys at the same timeline location as you manipulate the component entities in the level.

You can also use the Start Auto Recording option. When you choose this option, the playhead automatically moves along the timeline at the specified time. The default is one second. This means that the playhead skips ahead every one second. You can specify a lower value, which means the playhead sets keys more frequently. However, this can result in many keys that you must modify after you are finished recording.

To start auto recording

1. In the Track View, select or create the sequence that you want.
2. In the Play toolbar, click the Start Auto Recording icon. The playhead automatically moves in the timeline based on the setting specified in the drop-down menu.
3. In the viewport, manipulate a component entity that is part of your sequence. As the playhead moves and you make changes to component entities, your changes appear in the timeline as animation keys.
4. To stop auto recording, click the Start Auto Recording icon again.

Example 1: Recording Component Entity Changes

The following example uses the Transform component, but the same idea applies to any component tracks that are specified as animation keys in the sequence. In record mode, you don't need to manually set keys each time you make changes to your component entities.

1. In a level, you have a component entity that is part of a sequence.
2. By default, the Transform component on the component entity already has tracks in the sequence for the Position and Rotation properties.

3. To maintain the current position and rotation of the game object, add animation keys for both tracks at 0 on the timeline.

4. Select and drag the playhead to a different location on the timeline. This example moves the playhead to 1 second.

5. Move the Track View window to the side or dock it so that it's still open but not blocking the viewport for the currently open level.

6. Click the Start Animation Recording icon to enter record mode.

7. In the viewport or the Entity Outliner, select the component entity.

8. Use the Transform tool to move the entity farther back into the level. You can also enter different values in the Transform component properties.


10. Move the playhead to where you want to set another animation key and then adjust the property values again.

11. When finished, click the Start Animation Recording icon again to exit record mode.

Example 2: Recording Camera Movement from Camera View

The following example shows how you can add animation keys automatically in the sequence as you position your camera within the level from the perspective of the specified camera.

1. In a level, you have a component entity with the Camera component that is part of your sequence.
2. By default, the **Transform** component on the component entity has tracks in the sequence for the **Position** and **Rotation** properties.

3. To maintain the current position and rotation of the game object, specify the animation keys on both tracks at 0 on the timeline.

4. Move the playhead to a different location on the timeline.

5. Move the Track View window to the side or dock it so that it's still open but not blocking the viewport for the currently open level.

6. Click the **Start Animation Recording** icon to enter record mode.

7. In the left-corner of the viewport window, right-click **Perspective** and choose **Camera** and then specify the camera that is part of your sequence. The text will change from **Perspective** to **Camera entity: NameOfYourCamera**.

8. While you are viewing from the perspective of the camera, when you move in the level, the camera will move as well. This includes any rotations. In record mode, this adds animation keys at the current playhead location.

9. Move the playhead when you need to set another animation key in the timeline and then move the camera view around in the level.

10. When finished, click the **Start Animation Recording** icon again to exit record mode.
Using Animation Curves

The Curve Editor displays animations as function curves. Each track's curves represent an animation of a property value, such as anchor, offset, color, or any property of a UI element.

To open the Curve Editor

1. Do one of the following:
   - In the Track View, choose View, Curve Editor.
   - In the View toolbar, choose the Curve Editor icon.
2. Select a keyframe in the timeline to view it in the Curve Editor.

Note
You can have the Track View and Curve Editor open simultaneously if you prefer to access both tools.

A curve has the following three parts:

1. Curve or spline.
2. Keys on the curve/spline.
3. Tangent handles for the keys.
To edit elements in a curve

1. Select a key to see the associated tangent handles and then drag the boxes on the keys or the ends of the tangent handles (including unify tangents and automatic tangents) to manipulate them.
2. When moving keyframes, press and hold Shift to constrain the movement to time only.
3. Press and hold Alt to scale the selected keyframes around the playhead location.

The path of the curve represents the transition of the value between the keyframes. If the value changes in a straight line between each keyframe (linear), transitions between keyframes aren’t smooth. The default curve causes the value to smoothly ease in and ease out. Each key has an in tangent and an out tangent. Depending on the preferred effect, you can use the toolbar icons to switch the tangents to auto, zero, step, or linear. You can also manually drag the tangent handles.

By default, animation tracks are recorded with a smooth transition. You can use the buttons in the toolbar at the top of the Curve Editor to change how the curves behave on either side of the selected key. You can also drag spline keys to a different point in the timeline.

See the following tips for working in the Curve Editor:

- To zoom in or out, scroll the mouse wheel
- To pan the view, click and drag the middle mouse wheel
- To select multiple spline keys, click and drag to select the keys

To adjust a spline key

1. In the Node Pane, select a track. The curves for that track appear in the Curve Editor.
2. In the Curve Editor, select a spline key.
3. Do one of the following:
   - Drag the spline key to a different point on the timeline.
   - Use the toolbar buttons to select a preset: auto, zero, step, or linear.

To edit multiple elements at once

1. In the Node browser, select the parent track or sub-track.
2. Drag the spline key to a different point on the timeline.
3. Use the toolbar buttons to select a preset: auto, zero, step, or linear.

Populating a Scene

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cinematic scenes, also known as sequences, consist of multiple nodes, tracks, and track events. When you create a sequence, the sequence name is always the top (parent) node in the node browser.

For more information, see Using the Track View Editor (p. 1584).

Topics

- Setting Sequence Properties (p. 1609)
- Working with Slices and Sequences (p. 1611)
- Adding Component Entities (p. 1613)
- Moving a Component Entity in a Sequence (p. 1620)
Setting Sequence Properties

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can set properties in the Track View to control how the track view sequence appears. For example, you can specify Autostart so that the track view sequence starts playing as soon as the game starts.

To set sequence properties

1. In the Track View, select the applicable sequence and click the Edit Sequence icon.
2. In the Sequence Properties dialog box, you can specify the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autostart</td>
<td>Plays the scene on game start.</td>
<td>Use for testing purposes only. For scenes that must always play on game start, use triggers instead.</td>
</tr>
<tr>
<td>NoSeek</td>
<td>Disables random seeks in a scene, such as jumping to a certain time.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>When to use</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Cut-Scene</strong></td>
<td>Enables various scene toggles. When selected, the following options are available:</td>
<td>This property is required for all scenes that are camera-controlled.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Non-Skippable</strong> – Sets the IAnimSequence::eSeqFlags_NoAbort bit flag to 1. This flag provides a UI to specify whether the sequence can be skipped. Your game code must implement the input and code required to skip a sequence. You can use the IAnimSequence::GetFlags() method in C++ to retrieve the set if bit flags for the sequence.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Disable Player</strong> – Disables the player (required for all camera-controlled scenes).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Disable Sound</strong> – Disables all sounds that are not in the scene.</td>
<td></td>
</tr>
<tr>
<td><strong>Update Movie System First</strong></td>
<td>Reverses the update order. Typically, the movie system updates after the entity system updates.</td>
<td>Fix bone-attached entities that lag behind the parent movement. This problem typically occurs if the parent locator position is animated in the Track View.</td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td>For <strong>Start Time</strong> and <strong>End Time</strong>, specifies the start and end times of the sequence.</td>
<td></td>
</tr>
<tr>
<td><strong>Display Start/End Time As:</strong></td>
<td>Displays the start and end times. You can specify one of the following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Frames</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Seconds</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This is a display-only option; the times are always stored in seconds.</td>
<td></td>
</tr>
<tr>
<td><strong>Out of Range</strong></td>
<td>Changes the movie time behavior when it passes the end of the sequence:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Once</strong> – The movie time continues past the end of the sequence.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Constant</strong> – The movie time is held at the end of the sequence.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Loop</strong> – The movie time loops back to the beginning of the sequence when it reaches the end.</td>
<td></td>
</tr>
</tbody>
</table>
### Property Details

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move/Scale Keys</td>
<td>Scales animation curves over the timeline when the start or end times are changed.</td>
<td>Use when you want to lengthen or shorten a sequence, and either slow down or speed up the animations to fill the same relative percentage of the sequence timeline.</td>
</tr>
</tbody>
</table>

3. When you are finished, click **OK**.

## Working with Slices and Sequences

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you create a track view sequence that has entities contained in slices or add a track view sequence in a slice, the process is the same as creating a track view sequence in a level. The workflow is identical. However, see the following best practices when working with slices and track view sequences.

For more information about slices, see Working with Slices (p. 510).

### Avoid External References

As with any reference in a slice, if you make an external reference from within the slice, that reference will break if you also instantiate the slice in another level.

For example, you have a track view sequence that is in a slice and also animates an entity outside of the slice. When you instantiate that slice in another level, the track view sequence will not be able to find the external animated entity. The animation for that specific entity will not apply to any entity.

To avoid this, ensure that entities that you animate from a track view sequence in a slice is either in the same slice or in a child slice.

### Determine Which Instantiated Slice is in the Active Track View Sequence

If you instantiate a slice that has a track view sequence, this creates another track view sequence with the same name as the original slice.

**Example: Determine which sequence is active**

1. You create a track view sequence named `Seq` that is in a slice and you then instantiate that slice three times in the level.
2. In the Track View, for the **Active Sequence** drop-down menu, you will see `Seq` three times, one for each instantiated slice. Because the slices have the same name, it's unclear which instance of the sequence that you are editing in the Track View.
3. To determine which slice you are editing, in the Track View, in the **Node Browser**, right-click the track view sequence and in the context menu, choose **Select in Viewport**.
4. In the **Entity Outliner**, you can see the selected slice instance and track view sequence that you are editing.

### Avoid Animating the Parent Entity in a Slice

If you instantiate a slice for a track view sequence, ensure that parent entities are not animated. You can animate child entities, but the parent entity cannot be animated in a slice.
Example: Animate Child Entities in a Slice

The Entity Outliner shows two Parent slices, which have two children entities and a track view sequence.

In the Track View, the ExampleSequence animates only the two children entities in the slice. The Parent entity cannot be animated in a slice.

Note
The Transform (p. 878) component is animated in local space relative to its parent. When a slice is instantiated and its root transform is moved, all the child entity animations within also move as well. If you instantiate the slice and move it to a new location, the animations will play at this new location.

Avoid Setting a Director's Camera Track in a Sequence in a Slice

You should assume that multiple slices might be instantiated and playing back the slice sequence at the same time. Therefore, if you have a Director node that sets the camera track in a sequence, that
track may be playing at different local movie times simultaneously. Also, the order of slice animation playback may change between ticks. As a best practice, do not use the Camera track in Director nodes for sequences in slices.

For more information, see Director (Scene) Node (p. 1591).

Adding Component Entities

In the Track View, sequences determine what to animate based on what you add to the sequence. For example, if you want to animate an entity with an Actor or Light component, you can add that component entity to a sequence and then modify its properties.

To add a component entity to a sequence
1. In the viewport or the Entity Outliner, select the entity.
2. In the Track View, select or create the sequence that you want.
3. Do one of the following:
   • In the node browser, right-click and choose Add Selected Entity.
   • On the Sequence/Node toolbar, click the Add Selected Entity icon.

All component entities have the Transform component. This means you can animate the Position and Rotation properties for each component entity that is part of the sequence as needed.

If there are other components attached to the entity, those may also appear in the sequence. However, some components are not available for animating in the Track View.

For components that are available, you can right-click them to see what additional tracks can be added to the sequence. Some component properties can't be added as a track in a sequence and can't be animated.

For more information, see Component Entities and Component Nodes (p. 1586).

See the following topics for working with common component entities in a sequence.

Topics
• Adding Lighting (p. 1613)
• Adding a Camera (p. 1615)

Adding Lighting
You can set up a different lighting scenario in a sequence using light components and/or a time of day settings that are only triggered during the sequence. You can add light components to the sequence and then add the tracks that you want to animate. You can then set the Console Variable Node (p. 1590) to specify the time of day settings.

**Cinematic Lighting Best Practices**

See the following recommended guidelines and best practices for cinematics lighting.

- Lights should be turned on and off while in the Track View. If lights are off by default, they won’t accidentally render in game or interfere with a scene shot. You can animate the Visible track for each light to determine when a light should be turned on or off.
- Disable gameplay and cubemap lights as needed for shots to avoid interference.
- For pre-rendered cinematic scenes, use the console variable `e_timeofday` to trigger the correct time of day.

For more information, see Using the Console Window (p. 210).

- For real-time cinematics, use a track event node to trigger the correct time of day. For more information, see Event Node (p. 1595).
- For pre-rendered cinematic scenes, use the Shadows Setup Node (p. 1598) to enable high quality shadows mode.
- For pre-rendered cinematic scenes, because performance isn't an issue, you should always enable shadow casting and use as many spotlights as needed. Projector textures should be used as much as possible for spotlights. The SpecularMultiplier value should always be 1.
- Shadowmap quality from point lights improves greatly when the ProjectorFOV value is as low as possible. To soften shadows, you can increase the ProjectorFOV value slightly, but this also decreases the accuracy of the shadowmap.
- Don’t use ambient lights as they can weaken contrast and illuminate unwanted areas. Instead, use cubemaps to make the deepest shadow as dark as possible, and then add lights to increase the overall illumination.

**Light Components and Exposed Tracks**

See the following light components and their properties that you can add as tracks to animate in a sequence.

**Area Light (p. 546) component**

- Ambient
- Area FOV
- Area Height
- Area Max Distance
- Area Width
- Color
- Diffuse Multiplier
- Specular Multiplier
- Visible

**Environment Probe (p. 587) component**

- Color
Populating a Scene

- Diffuse Multiplier
- Probe Area Dimensions
- Probe Attenuation Fallout
- Probe Box Height
- Probe Box Length
- Probe Box Projected
- Probe Box Width
- Probe Fade
- Probe Sort Priority
- Specular Multiplier
- Visible

Point Light (p. 706) component
- Ambient
- Color
- Diffuse Multiplier
- Point Attenuation Bulb Size
- Point Max Distance
- Specular Multiplier
- Visible

Projector Light (p. 723) component
- Ambient
- Color
- Diffuse Multiplier
- Projector Attenuation Bulb Size
- Projector FOV
- Projector Max Distance
- Projector Near Plane
- Specular Multiplier
- Visible

Adding a Camera

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add a camera to a sequence so that it presents the sequence from a particular point of view. This helps you control what is being displayed in the animation and how.

Some sequences may require multiple cameras to cut between for different shots. You can use any number of component entity cameras as part of your sequence. The Director (Scene) Node (p. 1591)
helps you determine your active cameras for the sequence as well. You can also set the depth of field (DoF) within the sequence to control the focus view of the camera for the sequence.

See the following topics for working with cameras.

**Topics**
- Adding a Director (Scene) Node (p. 1616)
- Positioning a Camera (p. 1617)
- Animating Depth of Field (p. 1619)

**Adding a Director (Scene) Node**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To determine what camera you want your sequence to use, you must add a Director (Scene) Node (p. 1591).

**To add a Director node in the Track View**

1. In Lumberyard Editor, choose **Tools, Track View**.
2. In the Track View, select or create a sequence.
3. Do one of the following:
   - Right-click your sequence and choose **Add Director (Scene) Node**.
   - Click the **Add Director Node** icon.

By default, the Camera track appears on the Director (Scene) Node to control which camera is active for the sequence. This is the only track that is needed to assign and switch between cameras.

For more information about other tracks that you can add, see the Director (Scene) Node (p. 1591).

When you add an animation key on the Camera track, you can select the key and assign which camera should be active from the Key Properties pane. When a camera is set, you can see the name of the camera on the track.

To create a series of jump cuts from one camera to another camera in a sequence, add animation keys on the Camera track for the Director node. The Director node uses the key properties set for the Camera track to determine which camera to switch.
Positioning a Camera

You can control a camera's position and rotation to manipulate the facing direction during a sequence. These can be animated over time to make the camera dynamic. If you have multiple cameras, you can also control if the cameras switch immediately or if they should blend over time.

Animating the Camera

As a recommended workflow, enter record mode and then animate the camera while viewing from the specific camera's perspective. For more information, see Using Record Mode (p. 1604).

Note
You can still manipulate the component entity camera within the level from the default editor camera. However, this method makes it more difficult to determine where the camera is facing.

Example

1. In a level, you have a component entity with a Camera component attached. This component entity is part of your sequence.
2. By default, the Transform component attached to the entity has the Position and Rotation tracks in the sequence.
3. To maintain the current position and rotation of the game object, add animation keys on both tracks at 0 on the timeline.
4. Select and drag the playhead to a different location on the timeline.
5. Move the Track View window to the side or dock it so that it's open but not blocking the viewport for the currently open level.
6. Click the Start Animation Recording icon to enter record mode.
7. In the left-corner of the viewport window, right-click Perspective and choose Camera and then specify the camera that is part of your sequence. The text will change from Perspective to Camera entity: NameOfYourCamera.
8. While you are viewing from the perspective of the camera, as you move in the level, the camera moves as well. This includes any rotations. In record mode, this adds animation keys at the current playhead location.
9. Move the playhead when you need to add another animation key in the timeline and then move the camera view around in the level.

10. When finished, Click the **Start Animation Recording** icon again to exit record mode.

### Blending Cameras

By default, when you switch between cameras, the change is immediate. However, if you want a smoother transition, you can blend the cameras together. You can blend a camera in and out of the camera-controlled sequence, or blend between cameras that are part of the sequence.

#### Blending from Game Camera to Sequence Camera

**To use the default game camera**

1. Under the **Director** node, for the **Camera** track, add an animation key to the timeline.
2. Double-click the animation key and verify that the **Camera** property is set to **None**.
3. Add another key on the timeline for the **Camera** track. You can add the key at the beginning or end of the sequence, or both.
4. Double-click the animation key before a camera change, and for **Blend time**, specify a value in seconds. This value determines how the current camera will blend into the next.

   **Note**
   
   This creates a blend between the game camera to the sequence camera, or from the sequence camera back to the game camera, depending on where you place the key and adjust the **Blend time**.

#### Blending Cameras within a Sequence

Blended camera keys will blend the position, rotation, and field of view of the current camera into the next camera on the **Camera** track. This allows the cut to appear as a continuous single camera motion rather than an abrupt jump cut.

**To create a blended camera key**

1. Select the key for the first camera of the blend.
2. Double-click the key and do the following:
   
   a. For **Camera**, select the camera that you want to start.
   b. For **Blend time**, specify a value greater than 0. This is the time in seconds over which the blend will occur.
Example

If you add the first camera as a track view node, then you must add at least one animation key for the Position, Rotation, and Field of View tracks when using blended sequence cameras.

In this sequence, the Camera track starts at 0 seconds for CinematicsCamera and will blend into Camera5 at 2 seconds.

Animating Depth of Field

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Camera focus, or depth of field (DoF), is used to add realism to scenes, which simulates the way a real-world camera works. You can use a broad DoF to focus on the entire scene, or use a shallow DoF to have sharp focus only on objects that are a specific distance from the camera.

See the following guidelines and best practices when setting up camera focus:

- Always keep characters in focus.
- Shift focus slowly and deliberately.
- Don't overdo camera focus
- Don't use DoF for scenes that are far away. DoF works best for differentiating between closeups and the background.

- Use your eyes to focus at different distances and see what is sharp and what is blurred. You can use your thumb as a helper. This should give you a sense of how it should look in a scene.

DoF is rendered only for a single view pane layout (the default) in the viewport in Lumberyard Editor. If you are using a multiple view pane layout and the sequence camera is not in the active pane, DoF doesn't render. If you need to set this, do the following.

To set the viewport for a single view pane layout

1. In Lumberyard Editor, right-click the Perspective title bar in the viewport and choose Configure Layout
2. In the Layout Configuration dialog box, select the single view pane and then click OK.
3. Right-click the Perspective title bar again and choose Sequence Camera.

To add a Depth of Field node

1. In the Track View, select or create the sequence that you want.
2. In the node browser, right-click the Director node or any Camera node and choose Add Depth of Field Node.

Camera nodes take precedence over the Director node. Use the Director DoF node if you want the same DoF setup for multiple cameras. In most cases, you want separate, specific DoF setups for each camera for more control.

You can add as many keys as you want, and use the Curve Editor to further adjust DoF settings to change over time.

For more information, see Using Animation Curves (p. 1607).

Moving a Component Entity in a Sequence

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Each component entity has a Transform component. By default, when you add a component entity to a sequence, the Position and Rotation tracks appear automatically. You can manipulate the Position and Rotation properties for the component entity in a sequence. You can also add the Scale track if you want to manipulate the overall size of the component entity.

Each of the tracks has a subtrack for XYZ properties so that you can animate each axis.
You can add any number of keys for each track as needed, and adjust the transitions with the Curve Editor.

For more information, see Using Animation Curves (p. 1607).

When moving component entities, we recommend the following workflows:

- Manipulate the component entity in the level and then manually add your animation keys.
  
  For more information, see Adding and Removing Animation Keys on Tracks (p. 1601)
- Use record mode to set your keys. For more information, see Using Record Mode (p. 1604).

To set transform keys with record mode

1. In the Track View, create or select sequence and add a component entity. See Adding Component Entities (p. 1613).
2. By default, the Transform component attached to the entity automatically adds the Position and Rotation tracks to the sequence.
3. To maintain the current position and rotation of the game object, add an animation key for both tracks at 0 on the timeline.
4. Select and drag the playhead to a different position on the timeline.
5. Move the Track View window to the side or dock it so that it’s still open, but not blocking the viewport for the currently open level.
6. Click the Start Animation Recording icon to enter record mode.
7. In the viewport or the Entity Outliner, select the component entity.
8. Use the translation tool to move the entity farther back into the level. You can also enter values in the Transform component properties.
9. View the timeline for the sequence. New keys appear on the **Position** track at the 1 second position.

10. Move the playhead to where you want to set another animation key and then adjust the property values again.

11. When finished, click the **Start Animation Recording** icon again to exit record mode.

### Animating Characters in Scenes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To create character animations with the Track View, you can add the **Simple Motion** component and the **Actor** component to an entity. This creates a character, which you then add to a track view sequence.

**Topics**
- Creating Character Animations with the Simple Motion Component (p. 1622)
- Moving an Entity in a Scene (p. 1629)

### Creating Character Animations with the Simple Motion Component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To create character animations with the Track View, you can add the **Simple Motion** component and the **Actor** component to an entity. You then add the entity to a track view sequence and specify the motions that you want your character to animate. When you add a motion track to a track view sequence, the Track View drives animation on the **Simple Motion** component and its properties:

- **Motion**
- **Play speed**
- **Play time**
- **Blend in time**
- **Blend out time**

For more information, see the **Simple Motion** (p. 826) component.
Note

- The Play speed property is always set to 0.0. This is because the Track View will set the Play time value every frame to drive the playback of the motion. This allows scrubbing and playback in the Track View as well as playback in the game.
- When you add the Simple Motion component to a track view sequence, the Preview in Editor property is automatically enabled.

The following procedure uses Starter Game (p. 148).

To add the Simple Motion component in the Track View editor

1. In Lumberyard Editor, right-click in the viewport and choose Create entity.
2. Enter a name for the entity.
3. In the Entity Inspector, click Add Component, and then choose the Simple Motion component.
4. Add the Actor (p. 536) component.
5. In the Actor component, for Actor asset, specify an actor file. For example, you can specify the Jack.fbx file.

Example

Your entity should look like the following.
6. In Lumberyard Editor, choose **Tools, Track View**.

7. Click the **Add Sequence** icon, enter a sequence name, and then click **OK**.

8. Select the entity in the viewport, right-click the sequence, and then choose **Add Selected Entity**.

This adds the entity and its components to the track view sequence.

9. Right-click **Simple Motion**, choose **Add Track**, and then choose the **Motion** option.

10. In the timeline, for **Simple Motion**, double-click to create to a key.

11. Click the key and in the **Key Properties** dialog box, specify the following:
   
   a. For **Motion**, click the folder icon and specify the motion file, such as `jack_idle_to_walk.motion`.
   
   b. Specify the parameters that you want, such as the **Start Time** and **End Time**.

   **Note**
   You can set the **Loop** parameter so that motion continues to play as long as the track view sequence is set.

**Example**

12. Click the play icon to view the animation in the track view sequence. You can also drag the Track View needle across the timeline.
### Example

The following shows the actor animating the idle to walk motion.

![Animation Track View](image)

#### Note

When you update properties in the Track View, the original values for the entity properties are overwritten and are not restored. For example, if you set the **Start time** to 3.0 in the Track View, this updates the **Simple Motion** component properties in the **Entity Inspector**. If you want to reuse an entity with a **Simple Motion** component after a track view sequence is completed, update the **Play speed** parameter; the **Play speed** parameter always resets to zero in the Track View. You can also avoid this issue by not reusing entities in this way.
Blending Motions in the Track View Editor

The **Simple Motion** component supports two motions at a time for blending: the currently playing motion and the previously played motion. For example, you can blend two motions so that your actor smoothly transitions between a walk motion to a run motion.

You can set the **Blend In Time** and **Blend Out Time** properties for simple animation blending.

- **Blend In Time** – Specifies how long it takes for the motion that is set to fully blend in from zero to one second.
- **Blend Out Time** – Specifies how long it will take for a motion to fully blend out from one to zero second.

To blend two motions, overlap two animations and set the **Blend Out Time** of the last motion to 0.33 seconds, and the **Blend In Time** of the next motion by 0.33 seconds. This allows the two motions to smoothly chain together in the track view sequence.

**Note**
- If you want your animation to start when the game starts, click the **Edit Sequence** icon and in the **Edit Sequence** dialog box, select **Autostart**, and then click **OK**. For more information, see Setting Sequence Properties (p. 1609).
- The **Blend In Time** and **Blend Out Time** parameters affect the bone weight that are set in your DCC.

To blend motions in the Track View Editor

1. Double-click the first key and for **Blend Out Time**, enter 0.33. This allows the first motion, `jack_idle_to_walk.motion`, to blend into the next motion.

**Example**

2. Double-click the timeline to create a second key.

3. Double-click the key again and in the **Key Properties** dialog box, specify the following:
   a. For **Motion**, click the folder icon and specify the next motion, such as the `jack_strafe_run_forwards.motion` file.
   b. Specify the parameters that you want, such as the **Start Time** and **End Time**. For **Blend In Time**, enter 0.33. This allows enough time to overlap with the previous motion.

**Example**
4. Select and drag the second key so that overlaps with the first motion track.

5. Repeat steps 2 to 4 to add additional motions. For a motion that is ending, enter 0.33 for **Blend Out Time**. For the motion that starts next, enter 0.33 for **Blend In Time**.

**Example**

The following is a timeline with four motions blended together.

6. Click the play icon to view the track view sequence. You can also drag the Track View needle across the timeline. The motions blend together in the track view sequence.
Moving an Entity in a Scene

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Track View to move or rotate any entity in a scene.

To move a character in a scene

1. In the Track View, add the character to the desired sequence, then click the red record button.
2. In the Lumberyard viewport, click the character, then move or rotate as desired. This automatically updates keys at the current position of the slider in the track view sequence timeline slider.
3. Double-click the key to access Key Properties, then adjust values as needed.
4. Click the Curve Editor button, then select the tracks where the curve needs to be adjusted.
5. Drag a selection box around all the keys you want to change.
6. Click the **Set In/Out Tangents to Auto** button.

## Capturing Image Frames

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can capture image frames using render output, a capture track, or console variables.

**Topics**
- Capturing Image Frames with Render Output (p. 1630)
- Capturing Image Frames using a Capture Track (p. 1630)
- Capturing Image Frames with Console Variables (p. 1631)

### Capturing Image Frames with Render Output

You can use the **Render Output** tool in the Track View to capture image frames.

**To capture image frames using Render Output**

1. In Lumberyard Editor, choose **Tools, Track View**.
2. In the Track View, click **Tools, Render Output**.
3. In **Render Output**, set the input and out properties, and then click **Add**. You will see the capture added under **Batch**.
4. Click **Start** to start the capture.

**Note**

You may need to adjust the aspect ratio for captured image frames.

**To change the aspect ratio for image frame captures**

1. In Lumberyard Editor, choose **Edit, Editor Settings, Global Preferences**.
2. In **Preferences**, click **Viewports**.
3. Under **General Viewport Settings**, change the value for **Perspective View Aspect Ratio**. The default value is 1.3333.

### Capturing Image Frames using a Capture Track

You can capture image frames when a sequence is played in game mode.

**To capture image frames using a capture track**

1. In Lumberyard Editor, choose **Tools, Track View**.
2. In the Track View, right-click the **Director** node and choose **Add Track, Capture**.
3. Double-click the created track to add a capture keyframe. You can set the following key properties:

<table>
<thead>
<tr>
<th>Key Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Duration</strong></td>
<td>Sets the capture duration in seconds.</td>
</tr>
</tbody>
</table>
Capturing Image Frames

### Key Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time Step</strong></td>
<td>For a fixed frame rate, in seconds, by using a specified time step. Time step = 1/number of frames. Units are in 1 fps (frames per second), so a time step value of 0.0333 results in a game frame rate of 30 fps.</td>
</tr>
<tr>
<td><strong>Output Format</strong></td>
<td>Specifies the image output file format.</td>
</tr>
<tr>
<td><strong>Output Prefix</strong></td>
<td>Specifies a prefix to apply to the image file names. For best results, use the same prefix as the sequence for clarity.</td>
</tr>
<tr>
<td><strong>Output Folder</strong></td>
<td>Specifies the directory where the image files are stored under the following path:</td>
</tr>
<tr>
<td></td>
<td>lumberyard_version\dev\Cache\project_name\pc\project_name\</td>
</tr>
<tr>
<td><strong>Buffer(s) to capture</strong></td>
<td>Specifies the image capture format. Choose from the following options:</td>
</tr>
<tr>
<td></td>
<td>• Color – RGB pixel information</td>
</tr>
<tr>
<td></td>
<td>• Color+Alpha – RGB pixel data and Alpha channel data</td>
</tr>
<tr>
<td></td>
<td>Note: To capture Alpha channel data, you must set Output Format to tga.</td>
</tr>
<tr>
<td><strong>Just 1 Frame</strong></td>
<td>Chooses between single or multi-frame image capture.</td>
</tr>
</tbody>
</table>

4. Set up a Script Canvas graph to play the sequence on game start.

### Capturing Image Frames with Console Variables

Capture image frames with the following console variables. For more information, see Using the Console Window (p. 210).

**Capture Image Frames Console Variables**

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed_time_step</td>
<td>Lowers the game speed to achieve a constant frame rate throughout the sequence. For example, a time step value of 0.04 specifies a 25 fps gameplay speed. Default value: 0.0</td>
</tr>
<tr>
<td>capture_frames</td>
<td>Enables frame capture, if the value is set to 1.</td>
</tr>
<tr>
<td>capture_file_format</td>
<td>Sets the output format for the images. Valid values: .jpg, .tga, .tif</td>
</tr>
<tr>
<td>capture_file_prefix</td>
<td>Sets a file name prefix to use for captured frames.</td>
</tr>
</tbody>
</table>
Debugging Cinematic Scenes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can specify the following console variables when profiling a scene. For more information, see Using the Console Window (p. 210).

**Cinematic Scene Console Variables**

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
</table>
| e_DisplayInfo    | Displays basic performance information. The console variable also displays a warning if you exceed texture streaming memory. | 0 = Off  
1 = On  
2 = Enhanced  
3 = Compact |
| p_profile_entities | Runs your scene and looks for fluctuations. You can use this console variable to identify which entities are causing large peaks. | 0 = Off (default)  
1 = On |
| r_Stats | Toggles render statistics. Specify a value of 6 to finds assets with large draw calls or excessive materials. For example, this can help you find where shadows can be disabled, and so on. | 0 = Off  
1 = Displays per-frame and global render statistics.  
2 = Displays shaders for selected object.  
3 = Displays the CPU times for render passes and video memory usage.  
4 = Displays CPU times for render passes only.  
5 = Displays occlusion query calls. |
<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>= Displays per-instance draw call count.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>= Displays information about the total number of instances and batches.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>= Displays information about cleared render targets.</td>
<td></td>
</tr>
<tr>
<td>Console Variable</td>
<td>Description</td>
<td>Valid Values</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>e_DebugDraw</code></td>
<td>Displays helpers with information for each object.</td>
<td>1 = Name of the used .cgf file, polycount, and used LOD.</td>
</tr>
<tr>
<td></td>
<td>Specify a value of 2 to display polycount.</td>
<td>2 = Displays color coded polygon count.</td>
</tr>
<tr>
<td></td>
<td>Specify a value of 3 to display the current level of detail (LOD) for the selected entity.</td>
<td>3 = Displays color coded LODs count. Flashing colors indicate a single LOD.</td>
</tr>
<tr>
<td></td>
<td>For example, in the console window you can enter the following command:</td>
<td>4 = Displays object texture memory usage.</td>
</tr>
<tr>
<td></td>
<td>`e_DebugDraw 2</td>
<td>3`</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = Displays ambient color.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 = Displays triangle count, number of render materials, and texture memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = Renders geometry with simple lines and triangles</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 = Displays occlusion amount.</td>
</tr>
<tr>
<td></td>
<td>NOTES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This can take a long time to calculate, depending on level size.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 = Displays exported helpers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 = Displays the debug gun. Select an object for more information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 = Displays streaming information (buffer sizes).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 = Displays physics proxy triangle count for each object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 = Displays the texture memory usage for character attachments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21 = Displays an animated object distance to the camera.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22 = Displays an object's current LOD vertex count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23 = Displays an object in red if it's casting a shadow.</td>
</tr>
</tbody>
</table>

Version 1.28
1634
<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Displays objects with 0 LOD with red text.</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Displays objects with 0 LOD with red text and objects with 1 LOD with blue text.</td>
<td></td>
</tr>
<tr>
<td>e_CameraFreeze</td>
<td>Locks your current view, so that you can look around without redrawing any elements. You can use this console variable to locate problems and fix them, such as object culling and LOD. The view frustum is drawn in a white frame.</td>
<td>0 = Off (default) 1 = On</td>
</tr>
</tbody>
</table>
Graphics and Rendering in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Working with graphics and rendering in Lumberyard

Learn about how to work with the graphics and rendering features of Amazon Lumberyard. This documentation covers working with shaders, materials, lighting and shadows, and post-processing effects in your Lumberyard game.

In this section:

<table>
<thead>
<tr>
<th>Topic area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shaders and materials (p. 1636)</td>
<td>Learn how to use shaders and materials to create realistic looking entities and scenes, and how to customize the look-and-feel of the graphics in your Lumberyard game.</td>
</tr>
<tr>
<td>Lighting and shadows (p. 1779)</td>
<td>Learn how to add lighting and shadowing effects to your Lumberyard game.</td>
</tr>
<tr>
<td>Screen and post-processing effects (p. 1811)</td>
<td>Learn how to add camera- and fog-based effects to your Lumberyard game, as well as anti-aliasing for improved visual output quality.</td>
</tr>
</tbody>
</table>

Working with shaders and materials

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

There is a close relationship between materials, textures and shaders. For a material, you select a shader and then specify the material's properties and attributes such as color, specularity, and texture that are used by the shader for rendering the object. In this way, the shader entirely defines how the object looks. Lumberyard uses physically-based rendering (PBR) shaders, which use real-world physical rules and properties to describe how light interacts with the surface of objects. This means that game object...
materials look realistic under all lighting conditions. For more information, see Shader Rendering System (p. 1639).

For computer monitors, the sRGB (instead of RGB) color space is used. Using sRGB, you have greater precision for darker colors to which the human eye is more sensitive. sRGB also minimizes any banding artifacts. Always ensure that your monitor is calibrated properly. In sRGB, a 50% mid-gray is not 0.5 or 127 but rather 0.5 raised by the inverse of gamma 2.2, which equals 187 in Adobe Photoshop. For Photoshop, make sure that color management is be set to sRGB and Gray-to-Gray Gamma 2.2. By default, Gray is often set to Dot Gain 20%, which results in a color transformation in the alpha channel. A value of 127 comes into Lumberyard as 104 and cause inconsistencies.

The Material Editor is the primary tool used to create materials, texture mapping, setting opacity and lighting effects, setting shader parameters, vertex deformations, tessellation, and more, as shown below.

Topics

- Concepts (p. 1638)
- Shader Rendering System (p. 1639)
- Shader Reference (p. 1652)
- Selecting Material Surface Type (p. 1721)
- Setting Material Opacity (p. 1721)
- Setting Material Lighting and Color Settings (p. 1722)
- Material ID Mapping in Autodesk 3ds Max (p. 1723)
- Working with Textures (p. 1732)
Materials

A material has a set of properties that determines how its surface (p. 1721) reacts to physical actions, other materials, and its environment. For example, a metal surface is hard, doesn't shatter, reacts to bullets by generating spark particles, and has a unique sound when struck. Contrast this with a grass surface, which is soft, responds to wind, generates grass strands and dirt particles when hit, and sounds different than metal.

Within the definition of a material, you can also change its transparency (opacity (p. 1721)) level, color, specular reflections, glossiness, and glow (p. 1722). Textures (p. 1638) and shaders (p. 1639) are parameters of a material.

Textures

A texture is simply a 2D image. This image file can be a digital photo or other image file created with Photoshop, paint programs, or other applications.

In Lumberyard, these image files can be imported from .tif, .jpg, .png, .bmp, or .tga images. When you browse to select them in the Material Editor, for example, they appear with their original file extensions but are actually processed as .dds texture files.

Lumberyard uses textures to help define what the surface of an object looks like. You can layer textures on top of each other to define different aspects of the material's appearance. Lumberyard uses the following types of texture maps. These are the most common types you'll use with the default Illum (p. 1697) shader.

Diffuse (p. 1748)

Defines the base color of the material. For example, this could be a detailed image of a marble surface or wood grain. Defining a diffuse texture is optional; if you need a uniform color, you can instead define a diffuse color or tint.

Normal (p. 1749)

Simulates bumps on the surface of the material. Using a normal map can make a low-polygon mesh look like it has high-resolution geometry. Normal maps do not change the actual geometry of the surface to which it's mapped but only simulate variances in height. For example, a rock wall with a normal map applied might look like it has holes and projections that respond to light and shadows.
when the viewer faces it directly. But when viewed from the side, its silhouette appears as a flat surface.

**Specular**

Defines a surface's shininess and the color of its reflective highlights. A high value pixel (for example, white or light colors) renders a material as more shiny, such as metal details. A lower value pixel (for example, black, gray, or other dark colors) renders a less reflective surface, such as leather.

**Detail (p. 1751)**

Provides additional detail for close-up viewing. This map is typically a small image that is tiled many times across a larger surface. This level of detail would appear only at a close range; from a distance, the object still retains its normal level of detail. For example, a boulder can gain extra details such as scratches and dents or a porous surface when viewed at close range. Human skin can appear smooth from a distance but show pores and imperfections when viewed up close. To use, enable Detail Mapping under Shader Generation Params.

**Heightmap (p. 1757)**

Specifies height or elevation in a grayscale image that uses 256 shades. Black and white represent the lowest and highest elevations, respectively. High-quality 32-bit heightmaps (p. 1230) are recommended for large terrain areas. An important difference between heightmaps and normal mapping is that a heightmap changes the actual geometry of the surface to which you apply it. This is crucial, for example, if you apply a heightmap to a terrain, as objects and players need to respond to the changes in elevation. In Lumberyard, the heightmap setting in the Material Editor can be applied as offset bump map, POM, silhouette POM, and displacement mapping.

**Emittance Multiplier**

Multiplies the emissive color with RGB texture.

**Decal Opacity (p. 1752)**

Alpha mask used for decal entities. To use, enable Decal under Shader Generation Params.

**Emittance**

Multiplies the emissive color with RGB texture. Emissive alpha mask is contained in alpha channel.

**Shaders**

Shaders (p. 1639) use real-world physical rules and properties to describe how incoming light interacts with objects. This means that object materials look more convincing under different lighting conditions. The two main types of shaders are metallic and nonmetallic. Examples of metallic materials include iron, gold, and copper. Nonmetallic material examples include plastic, stone, wood, skin, and glass. A shader can also mix metal and nonmetal materials.

**Shader Rendering System**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard uses physically-based rendering (PBR) shaders that use real-world physical rules and properties to describe how incoming light interacts with objects. This means that object materials look more convincing under different lighting conditions. A basic understanding of how light interacts with objects in the real world can be very helpful when setting up materials.
Each shader has a unique set of shader parameters (Shader Params) and generation parameters (Shader Generation Params). Some shader parameters become available (are visible) only if an associated shader generation parameter is first enabled. This is also true for certain texture map slots (file paths) under Texture Maps. For a listing of all shaders, see Shader Reference (p. 1652).

There are two categories of materials that are relevant for shader rendering: metals such as like iron, gold, copper, and non-metals such as plastic, stone, wood, skin, glass. Each has different diffuse and specular reflectance characteristics.

**Shading Metallic Materials** - Metal reflects all visible light, hence has specular reflectance. The different types of metal have different specular colors, and should always be above sRGB 180. Metal has no diffuse reflection and thus has a black diffuse color. Rusty metal however needs some diffuse color.

**Shading Nonmetallic Materials** - In contrast, non-metals have diffuse reflection with weak, monochromatic (gray) specular reflections. Most non-metals reflect only 2%-5% of the light as specular. The sRGB color range for most non-metal materials is between 40 and 60 and should never be above 80. A good clean diffuse map is required for non-metals.

As the variation is so little, it is often enough to use a constant specular color instead of a specular texture map.

**Shading Mixed Metal and Nonmetal Materials** - Materials that contain both metals and non-metals require a specular map, as metal has a much brighter specular color than non-metal. If a specular map is used, the specular color should be set to white (255/255/255) - as it gets multiplied with the values from the specular map and would otherwise lower the physical values from the map.

To access a shader

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the left tree pane, select a material to work with.
4. Locate shader-specific parameters under **Shader Params** and associated **Shader Generation Params**.

**Topics**

- Image-Based Lighting (p. 1640)
- Environment Probes and Cubemaps (p. 1641)
- Height Map Ambient Occlusion (p. 1641)
- Developing a Custom Shader (p. 1642)

**Image-Based Lighting**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Image-based lighting is a rendering technique where complex lighting is stored in an environment map that is projected onto a scene. In simple words, a light probe or environment map is just an image on a sphere.

If the range of the image colors is within some small defined range (0-255 for monitor displays), the image is LDR (low dynamic range). With HDR (high dynamic range) some rendering effects become more
apparent and correct (DOF, motion blur, bloom, dark materials, global illumination). Depending on the image and compression requirements, various texture formats can be useful.

Diffuse lighting can be approximated very well by diffuse-convolving an environment map, which can be stored as a cube map again. Because of bilinear filtering, the texture can be quite low resolution. Mip maps are not required and the result with mip maps can actually look worse as ordinary mip mapping on the GPU is computed for each 2x2 pixel block and 2x2 block artifacts can become noticeable.

Environment Probes and Cubemaps

Cube mapping uses the six faces of a cube as the texture for a material. The cube map is generated by projecting and then rendering the scene six times from a single viewpoint, one for each cube face. In this way, the local environment can be stored as either six square textures, or unfolded onto six regions of a single texture. This texture is used to store the image of the environment surrounding the object. Cube maps are useful for showing reflections, and are relatively small in size because reflections can be blurry as long as you are not simulating a mirror.

Cube maps control shadow color, ambient diffuse, and particle diffuse as well as reflections. They function as bounce lighting by taking the colors from the surroundings and applying them directly into the diffuse texture of materials inside their specified radius.

For information about using environment probes and cubemaps, see Environment Lighting (p. 1779)

Cube maps use image-based lighting. For more information see Image-Based Lighting (p. 1640).

Height Map Ambient Occlusion

Ambient occlusion (AO) is a technique used to calculate how exposed each point in a scene is to ambient lighting. The lighting at each point is a function of other geometry in the scene. For example, the interior of a building is more occluded and thus appears darker than the outside of the building that is more exposed.

Lumberyard uses height map-based ambient occlusion (AO), which is a high-performance and efficient method of providing ambient occlusion in outdoor environments without the need for prebaking. This make it suitable for PC, consoles, and virtual reality headsets.

In combination with screen space directional occlusion (SSDO), height map AO provides additional shading cues that enhance the depth perception of a scene.

To enable height map ambient occlusion

1. In the Rollup Bar, click the Terrain tab, and then choose Environment.
2. **Under Terrain**, select the **Height map AO** check box.

The influence that height map AO provides can be restricted using clip volumes and vis areas. Both of these object types have a **IgnoreHeightMap AO** check box that will locally disable height map AO inside the volume or area.

By default, evaluation is performed at quarter-display resolution. This can be changed using the **r_HeightMapAO** console variable, as listed below.

Heightmap AO uses the following console variables:

- **r_HeightMapAO** – Sets the resolution that evaluation is performed at. Values are: 0=off, 1=quarter resolution, 2=half resolution, 3=full resolution.
- **r_HeightMapAOAmount** – Sets the strength of the occlusion effect when combined with the scene.
- **r_HeightMapAORange** – Area around the viewer that is affected by height map AO.
- **r_HeightMapAOResolution** – Texture resolution of the height map used for approximating the scene.

### Developing a Custom Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Most visual effects in Lumberyard are produced by shaders, which employ a number of standard and advanced lighting models like Blinn, Cook-Torrance, Oren-Naye, Kajiya-Kay, and some custom models.

There are two types of shaders used: lighting shaders that interact with scene illumination, and regular shaders that don’t calculate any lighting information and are used for post-processing effects. All lighting shaders have a common structure and make use of a unified shading interface. This interface should always be used to ensure proper usage of the lighting pipeline, minimize code duplication and save a lot of work.

Lumberyard uses an ubershader system with compile-time defines to handle the many different shader permutations that are required for combining numerous shader features. The shader format used that is very similar to High-Level Shader Language (HLSL), DirectX FX, and CgFX.

Shader development is a programming discipline onto itself and requires expert knowledge to optimize as shader code can be performance-critical and platform-dependent.

The easiest way to create new shaders is with a text editor.

**To create a new shader**

1. Using a text editor, copy an existing .ext extension file and associated .cfx effect file.
2. Add the shader's file name to the **lumberyard_version/dev/Editor/Materials/ShaderList.xml** file.
3. Restart the Material Editor. The new shader appears and you can assign it to a material.

### Topics

- Shader Development Best Practices (p. 1643)
- Shader Rendering Pipeline (p. 1644)
Shader Rendering System

- Hot Reloading of Shaders (p. 1644)
- Remote Shader Compiler (p. 1645)
- Generating Shader Combinations (p. 1651)

Shader Development Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Shaders add flexibility to the modern rendering effects seen in games today. To save you the trouble of managing multiple shader permutations, the shader compiler automatically creates the permutations for you when it parses the code, hiding the complexity. However, this process requires high memory performance and requires long compile times.

See the following guidelines and best practices when you develop a custom shader for Lumberyard:

- Before creating a new shader, make sure that you can't reuse or parameterize one of the existing shaders.
- Precompute as much as possible. Place the shaders in either textures or in the vertex shader and pass the data to vertex interpolators.
- For performances reasons, try to reduce the usage of sincos (8 ALU), normalize (3 ALU), pow (3–9 ALU), divisions (3 ALU), and smoothstep because of their higher execution cost.
- Pack as much data as possible in each texture instead of doing multiple texture lookups. Texture lookups are expensive on consoles and older hardware.
- Shader code is compiled depending on permutation flags that can come from various sources, such as material and runtime flags. These flags are defined in the following files:
  - `runtime.ext` – This file contains all runtime flags that can be shared by all shaders. You can review the file to better understand how to add flags and apply to any newly added shader that requires an existing runtime flag.
  - `shader_name.ext` – These files define shader-specific flags that are exposed in the Material Editor.
- Having many flags leads to many shader permutations. For best results, try to keep the number of flags relatively small.

Compiling Shaders in Debug Mode

To debug shaders in applications such as RenderDoc, Nsight, or Pix, it's important to enable shader debugging information. By default, shaders are optimized and don't output potentially useful debugging information. You can use the `r_ShadersDebug` console variable to enable debug information when needed.

To enable shader debug compiler options

1. To ensure a clean compile, we recommend that you first delete your `lumberyard_version\dev\Cache\your_game\platform_type\user\cache\shaders` directory.
2. In a text editor, navigate to the `lumberyard_version\dev\system_windows_pc.cfg` file.
3. Add the following console variable to the file.

   `r_ShadersDebug=3`
The console variable will change the DX11 compile flags in order to disable optimizations (/Od) and enable debugging information (/Zi).

4. Save the file. Lumberyard will automatically compile new shaders.

**Note**
You can modify some console variables at runtime. However, we recommend that you add this console variable to the `system_windows_pc.cfg` file. For more information, see Using the Console Window (p. 210).

**Shader Rendering Pipeline**

Lumberyard has a fixed rendering pipeline that is set up in the renderer code. Lumberyard is almost fully deferred and only does forward for hair, eyes, glass, transparencies, and water reflections. Lumberyard makes use of two elements: effects that define parameterized shader code, and materials that customize the shader parameters for a specific mesh.

First, Lumberyard fills the off-screen buffers like reflection buffers and shadow maps. After that, it writes the scene depth to the frame buffer and additionally to a render target. Having access to scene depth is essential for some subsequent rendering steps like screen space ambient occlusion or fog rendering. After the depth is written, Lumberyard does the forward lighting. The shadow contributions are written in a separate step to a texture that combines the shadowing result from several light sources (deferred shadowing). Finally, translucent objects are drawn in a back-to-front order.

When Lumberyard tries to render an object it will first check if a compiled shader is available. If the shader is not available, Lumberyard will try to load it from the global shader cache. If the shader cannot be found in the cache, the rendering thread will issue a request to stream the shader in from disk and will block until the streaming load is complete. This can cause severe stalls due to the relatively long time needed to load data from disk.

**Hot Reloading of Shaders**

Lumberyard supports hot reloading of shaders, so whenever you modify and save a shader file, it will get reloaded automatically and the results can be viewed directly in a test level.

For hot reloading to work, shader files must be copied to the appropriate locations, and the following requirements must also be met:

- Add the following code to the `dev\system.cfg` file:

```plaintext
sys_PakPriority=0 <!--ensures the shader files get loaded from the file system instead of from pak files>

r_ShadersEditing=1 <!--ensures that shader code can be recompiled at runtime-->
```

- In the Console, enter `r_reloadshaders 1`. This is only required in the game executable. In Lumberyard Editor, it will automatically reload a shader when you modify it.
For Lumberyard, copy the shader files to the `dev\Lumberyard\Shaders` directory.

**Remote Shader Compiler**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard's remote shader compiler application provides a convenient way to compile shaders during development. You can install the shader compiler on a PC or Mac local network server that can communicate over TCP. The server receives the shader source file from a computer running Lumberyard, compiles it, and sends back the shader. Almost all renderers require a remote shader compiler. The exception is a PC running DX11, for which a remote shader compiler is optional for local development.

The remote shader compiler is used to store all the shader combinations that have been requested by the game so far. Ideally, during development, you set up a remote shader compiler server for all designers, developers, and testers to connect to while building the game. This builds a comprehensive shader list (`ShaderList.txt`), which you then use to generate the shaders and compress them into `.pak` files for the game to use.

If preferred, you can also set up the shader compiler locally on a PC rather than setting up a central remote shader compile server.

Lumberyard also generates its own `ShaderList.txt`, which records all of the shader combinations requested on that particular instance of Lumberyard. When preparing the game for release, Lumberyard's local `ShaderList.txt` for shader generation is sufficient if the game is developed solely on one instance of Lumberyard. Otherwise, if a team is developing multiple levels or using a very large map, some shaders may be missed by using the `ShaderList.txt` in the Lumberyard application's file system.

On a PC, the remote shader compiler can generate shaders for Lumberyard running on the following systems. For information about the options listed with the systems, see Game Development Configuration (p. 1648).

- PC (DX11) with `r_ShadersRemoteCompiler=1`
- Consoles
- Android OpenGL ES 3.1 mobile device with `r_ShadersUseLLVMDirectXCompiler=0`
- Android OpenGL ES 3.0 mobile device with `r_ShadersUseLLVMDirectXCompiler=0`
- Mac with `r_ShadersUseLLVMDirectXCompiler=0`
- Mac with `r_ShadersUseLLVMDirectXCompiler=1`
- iOS mobile device with `r_ShadersUseLLVMDirectXCompiler=0`
- iOS mobile device with `r_ShadersUseLLVMDirectXCompiler=1`

On a Mac, the remote shader compiler can generate shaders for Lumberyard running on the following:

- Mac with `r_ShadersUseLLVMDirectXCompiler=1`
- iOS mobile device with `r_ShadersUseLLVMDirectXCompiler=1`

**Important**

Ensure the server or computer that is running the remote shader compiler is in a controlled environment that restricts incoming network requests to only authorized and trusted users or devices. Don't run the remote shader compiler with escalated root, admin, or super-user privileges.
Running the Remote Shader Compiler

You can find the remote shader compiler, CrySCompileServer, in the following directories:

- PC – \lumberyard_version\dev\Tools\CrySCompileServer\x64\profile\n- Mac – \lumberyard_version\dev\Tools\CrySCompileServer\osx\profile\

A configuration file (p. 1646) is also available for configuring the TCP port that the server application will listen on.

You can launch the remote shader compiler by starting CrySCompileServer.exe manually. However, usually it makes sense to set it up as a service, so that it always starts with the operating system.

Because requests for shaders are executed in parallel, you may notice significant delays in acquiring shaders at runtime.

Remote Shader Compiler Configuration

You configure the remote shader compiler by creating or editing the config.ini file, which is located in the following directories:

- PC: \lumberyard_version\dev\Tools\CrySCompileServer\x64\profile
- Mac: \lumberyard_version\dev\Tools\CrySCompileServer\osx\profile

To configure the remote shader compiler, edit the following parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailError</td>
<td>Set to an internal company email address to which notifications about compilation errors will be sent.</td>
</tr>
<tr>
<td>TempDir</td>
<td>The cache directory in which the binary shaders are stored once they are compiled. This parameter must point to a valid absolute path. The default path is C:\SHADER_CACHE.</td>
</tr>
<tr>
<td>port</td>
<td>TCP port, which must match the setting in the game system_platform_shader_version.cfg file. For examples, see the system_windows_pc.cfg, system_osx_metal.cfg, or system_android_es3.cfg file.</td>
</tr>
<tr>
<td>MailServer</td>
<td>Your email server.</td>
</tr>
<tr>
<td>SCMailAddress</td>
<td>Email address used in the From field of the email sent by the remote shader compiler.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PrintCommands</td>
<td>If set to 1 (enabled), the remote shader compiler prints out the commands it uses to generate shaders.</td>
</tr>
<tr>
<td>PrintWarnings</td>
<td>If set to 1 (enabled), the remote shader compiler prints out all the compilation warnings when shaders are generated.</td>
</tr>
<tr>
<td>white_list</td>
<td>If set to 0.0.0.0/0, the remote shader compiler authorizes any IP. Any device can connect to the remote shader compiler. As this can be a security risk, use this parameter with caution.</td>
</tr>
<tr>
<td>DumpShaders</td>
<td>If set to 1 (enabled), dumps the output of HLSL, GLSL, and metal shaders. This outputs shaders into the Shaders/platform-compiler-shader_language directory, where:</td>
</tr>
</tbody>
</table>
|                 | • **platform** = PC, Mac, iOS, or Android  
|                 | • **compiler** = D3D11_FXC, GLSL_HLSL, METAL_HLSLcc, or METAL_LLVM_DXC  
|                 | • **shader_language** = D3D11, GL_4, GLES3_1, GLES3_0, or METAL |

**Example**

The completed `config.ini` file can look like the following.

```
MailError = shadererror@your_company.tld
MailInterval = 1
port = 61453
TempDir = C:\SHADER_CACHE
MailServer = your_email_server
SCMailAddress = RemoteShaderCompiler@your_company.tld
PrintWarnings = 1
```

**Creating an allow list for the Remote Shader Compiler**

You can use an allow list to specify the IP addresses that are allowed to connect to your remote shader compiler. If a computer has an IP address that is not in the allow list, the remote shader compiler provides a message that an invalid computer tried to connect and then closes the connection. This prevents data from being read or sent to an invalid connection.

**To create an allow list for the remote shader compiler**

1. Create or edit the `config.ini` in the same directory as the remote shader compiler executable.
2. Add the following parameter to the file.

   `white_list` – Provide a comma-separated list of IP addresses in CIDR format. The remote shader compiler uses this list to validate incoming connection requests. The remote shader compiler automatically adds the loopback IP address (127.0.0.1) and its own IP address.

   **Note**

   To specify multiple IP addresses, specify a comma-delineated list. Do not include white space or empty lines.
Example

The following allows computers or devices with IP addresses matching the allow list to connect to the remote shader compiler.

```
white_list=10.53.180.37,10.53.104.220
```

The following allows computers or devices with an IP address of `192.168.0.*` to connect to the remote shader compiler. The `/24` specifies a net mask of 24-bits. If you specify `/8`, any address that starts with `192` is allowed, given only an 8-bit net mask.

```
white_list=192.168.0.1/24
```

Location of Specific Shader Compilers

In the root directory of the remote shader compiler, each shader language or render API has its own directory with additional subdirectories for different version numbers. They are located in the following directory:

```
lumberyard_version\Tools\CrySCompileServer\Compiler
```

You can find information about the path used by the remote shader compiler in the `ShaderCache.cpp` file, under the function `mfGetShaderCompileFlags`.

Lumberyard provides all appropriate shader compilers for you that match the code of that version.

Shader Cache Lists

The remote shader compiler contains different text files of all the combinations requested so far by the game. These files are important as the shader `.pak` files cannot be generated without them.

The shader cache list files are located in the following directories:

- **PC** – `lumberyard_version\dev\Tools\CrySCompileServer\x64\profile\Cache\project_name\platform-compiler-shader_language`
- **Mac** – `lumberyard_version\dev\Tools\CrySCompileServer\osx\profile\Cache\project_name\platform-compiler-shader_language`

Where:

- `platform` = PC, Mac, iOS, or Android
- `compiler` = D3D11_FXC, GLSL_HLSL, METAL_HLSLcc, or METAL_LLVM_DXC
- `shader_language` = D3D11, GL_4, GLES3_1, GLES3_0, or METAL

The game submits the requests to the remote shader compiler either during actual gameplay or during loading phases, even when remote shader compiling is disabled. This ensures that all possible shader combinations are collected and that the shader caches, which are generated during the shader cache generation phase, are as complete as possible.

Lumberyard also generates shaders locally. These text files are named `ShaderList_platform.txt` (for example, `ShaderList_DX11.txt`). You can find these files in the `lumberyard_version\dev\cache\project_name\pc\user\cache\shaders` directory.

Game Development Configuration

A remote shader compiler server can provide a performance benefit as it caches the results and sends them to team members instead of having to compile shaders each time. In addition, the server keeps
track of all shaders used by all people, which can be valuable if you want to make a release build that includes all shaders.

You can set the following parameters in the `system_platform_shader_version.cfg` file in Lumberyard's root directory, such as `system_android_es3.cfg` or `system_windows_pc.cfg`.

<table>
<thead>
<tr>
<th>Console Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r_ShadersUseLLVMDirectXCompiler=1</code></td>
<td>Turn on the shader compiler that generates metal shaders on Windows and Mac.</td>
</tr>
<tr>
<td><code>r_ShadersRemoteCompiler=1</code></td>
<td>Specifies whether the game enables or disables the remote shader compiler.</td>
</tr>
<tr>
<td></td>
<td>If set to 0, Lumberyard does not compile remote shaders and will instead compile local shaders.</td>
</tr>
<tr>
<td><code>r_ShaderCompilerServer=IPv4_of_PC_running_the_RemoteShaderCompiler</code></td>
<td>Specifies the remote shader compiler location. When the remote shader compiler is enabled, the game needs the location of the remote shader compiler.</td>
</tr>
<tr>
<td><code>r_ShaderCompilerServer=localhost</code></td>
<td>If you are running on a PC, specify this console variable to use the remote shader compiler locally.</td>
</tr>
<tr>
<td><code>r_ShaderCompilerServer=10.0.0.10;10.0.0.11</code></td>
<td>Specifies more than one remote shader compiler. Separate multiple IP addresses with semicolons. Note: It is not possible to use the network name of the server instead of the IP address, because name resolving is not performed.</td>
</tr>
<tr>
<td><code>r_ShaderCompilerPort=portnumber</code></td>
<td>If the remote shader compiler server uses a user-defined port number as specified in the <code>config.ini</code> file, you can configure the port number with this console variable.</td>
</tr>
<tr>
<td><code>r_shaderssubmitrequestline=0</code></td>
<td>Disables submit request lines to the remote shader compiler. This is console variable is useful when you are experimenting with shaders and you don't want to add these combinations to the shader cache.</td>
</tr>
<tr>
<td><code>r_AssetProcessorShaderCompiler=1</code></td>
<td>You can use Asset Processor to proxy remote requests to the shader compiler server if a device cannot connect to the shader compiler server. In this case, set <code>r_AssetProcessorShaderCompiler=1</code>. Now, whenever the game would have made a request directly to the shader compiler server, it instead submits the request to Asset Processor (this can also be over a USB connection), which then forwards it to the shader compiler server.</td>
</tr>
<tr>
<td><code>r_shadersAsyncCompiling=3</code></td>
<td>Allows shaders to stream asynchronously and prevents the game from freezing while waiting for the shaders to compile.</td>
</tr>
</tbody>
</table>
Console Variables | Description
--- | ---
r_ShadersAsyncActivation=1 | Allows shaders to stream asynchronously. Set to 0 to prevent the shaders from streaming asynchronously.

Creating Paks for Server Assets

Specify `RC.exe` to build a pak file that contains server assets only. You generate these assets by using the following command:

```
AssetProcessorBatch.exe /gamefolder=my_game /platforms=server /server)
```

You can update `RC.exe` to look for assets in the `lumberyard_version\dev\cache\MyGame\server` directory, instead of using the PC client assets.

Example

To create paks for server assets

1. Navigate to the `lumberyard_version\dev\Bin64vc141\rc` directory.
2. In a text editor, open the `RCJob_Generic_MakePaks.xml` file.
3. Make a copy of the file and name it `RCJob_Generic_MakePaks_Server.xml`.
4. Edit the file so that the `src` points to your server assets, instead of the PC client assets.

   Before

   ```xml
   # <DefaultProperties
   p="pc"
   game="samplesproject"
   src="cache\$(game)\$(p)"
   trg="$(game)_$(p)_paks"
   />
   ```

   After

   ```xml
   # <DefaultProperties
   p="pc"
   game="samplesproject"
   src="cache\$(game)\server"
   trg="$(game)_server_paks"
   />
   ```

5. Save the file.
6. Now when you run `RC.exe`, you can assign the job to the new xml file.

   **Example**

   ```
   rc.exe /job=path\to\RCJob_Generic_MakePaks_Server.xml /p=pc /game=my_game /trg=BinTemp \server_paks
   ```

**Generating Shader Combinations**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Make sure that the Remote Shader Compiler (p. 1645) has been set up successfully first. The remote shader compiler should be accessible by everyone playing the game, especially QA. Try to have everyone who is working on a certain game project share the same remote shader compiler.

Normal game builds should contain shader cache .pak files generated by the shader cache generation phase. At the beginning of a project this could be either completely missing (because no shaders requests have been submitted yet) or the .pak files could still be missing a lot of shaders.

When Lumberyard tries to render an object it will check if the compiled shader is available. When the shader is not available, it will try to load it from the global cache. This can either be loaded directly or through the streaming engine. The direct loading will cause direct disc access from the render thread and this could cause severe stalls due to the streaming thread trying to access the disc at the same time.

By default, when shader compiling is disabled, Lumberyard will stream the shaders from the global cache. The object won't be rendered when shader data is being streamed in. This default behavior can be modified with the following console variable. Note that streaming of shaders is not allowed when shader compiling is enabled, and Lumberyard will automatically disable the following console variable:

```
r_shadersAsyncActivation = 0
```

When the shader is missing from the global cache, a "request line" to store this missing shader is directly sent to the remote shader compiler to be sure that this shader will be available in the next shader cache generation. This happens even when shader compiling is disabled, but the remote shader compiler needs to be active.

When no shader compiler is defined or if the shader compiler is disabled then the request line will be ignored. It is recommended to test the remote shader compiler as much as possible to collect as many shader combinations as possible. The remote shader compiling can be disabled with the following console variable, which is disabled by default in release builds, otherwise is always enabled:

```
r_shadersRemoteCompiler = 0
```

The submission of the shader request lines can be disabled as well:

```
r_shadersSubmitRequestLine = 0
```

When shader compiling is disabled and the shader is missing in the global cache, the object won't be rendered at all. When shader compiling is enabled, and the remote shader compiler is active, an asynchronous request to compile the shader will be sent to the remote shader compiler. If the remote shader compiler is disabled, then the shader will be compiled locally on the PC platform. Other game platforms do not support local compilation.

To keep track of the current shader cache state in game, extra debug information can be enabled using the following console variable:
r_displayinfo = 2

A shader cache information line can be found on the top right of the screen, which reports the amount of Global Cache Misses (GCM) that have been found so far. It also reports if shader compiling is currently enabled or not.

All the shader cache misses also get written to a text file at the following location: \Shaders \ShaderCacheMisses.txt. This information is only used for debugging the current state of the shader cache, and should ideally be empty.

**Shader Reference**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes the following physically-based rendering (PBR) shaders, which use real-world physical rules and properties to describe how light interacts with the surface of objects. This means that game object materials look realistic under all lighting conditions.

**To access a shader**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the left tree pane, select a material to work with.
4. Locate shader-specific parameters under **Shader Params** and associated **Shader Generation Params**.

**Note**

Some shader parameters become available (are visible) only if an associated shader generation parameter is first enabled. This is also true for certain texture map slots (file paths) under **Texture Maps**.

<table>
<thead>
<tr>
<th>Shader Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common.Cloud Shader (p. 1653)</td>
<td>Use to render 3D clouds that use per-vertex gradient lighting and takes sun color, sky color, and viewer position into account.</td>
</tr>
<tr>
<td>DistanceClouds Shader (p. 1657)</td>
<td>Use to render cheap 2D clouds that are distantly placed in a sky scene.</td>
</tr>
<tr>
<td>Eye Shader (p. 1659)</td>
<td>Use to render realistic eyes that take sclera, cornea, iris, and eye moisture properties into account. Eyelash rendering is done using the Hair Shader (p. 1693).</td>
</tr>
<tr>
<td>GeometryBeam Shader (p. 1689)</td>
<td>Use to create volumetric light beams that feature dust and turbulence effects.</td>
</tr>
<tr>
<td>Glass Shader (p. 1691)</td>
<td>Use to render glass surfaces with various refractive, reflective, ripple, tint, and cracking effects.</td>
</tr>
<tr>
<td>Hair Shader (p. 1693)</td>
<td>Use to render all hair and fur, imparting different color, stranding, and animation effects. Use to render eyelashes and eyebrows along with the Eye Shader (p. 1659) for realistic eyes.</td>
</tr>
<tr>
<td>Shader Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HumanSkin Shader (p. 1696)</td>
<td>Use to render skin and its various physical properties including color, oiliness, pores, stubble, and wrinkles.</td>
</tr>
<tr>
<td>Illum Shader (p. 1697)</td>
<td>The most common shader - use to create an extremely wide variety of render effects.</td>
</tr>
<tr>
<td>LightBeam Shader (p. 1702)</td>
<td>Use to create volumetric light beams that feature fog and other atmospheric effects.</td>
</tr>
<tr>
<td>Monitor Shader</td>
<td>Use to create retro television screen effects such as grain, noise, chroma shift, and interlacing. Useful for in-game displays.</td>
</tr>
<tr>
<td>NoDraw Shader</td>
<td>Use mainly for physics proxies, this shader does not render selected geometry.</td>
</tr>
<tr>
<td>ParticleImposter Shader (p. 1704)</td>
<td>Use to create particle effects that are not affected by light and hence do not cast shadows or cause reflections.</td>
</tr>
<tr>
<td>Particles Shader (p. 1704)</td>
<td>Use to render particle effects for fire, smoke, lightning, sparks, and fog that are affected by light and as such cast shadows and cause reflections.</td>
</tr>
<tr>
<td>Sky Shader (p. 1708)</td>
<td>Use to render cheap static sky (SkyBox) effects.</td>
</tr>
<tr>
<td>SkyHDR Shader (p. 1708)</td>
<td>Use to render realistic dynamic sky effects that change based on time of day in the level.</td>
</tr>
<tr>
<td>TemplBeamProc Shader (p. 1709)</td>
<td>Use to create cheap fog-like effects for light beams.</td>
</tr>
<tr>
<td>Terrain.Layer Shader (p. 1710)</td>
<td>Use for painting and blending terrain texture layers in a level.</td>
</tr>
<tr>
<td>Vegetation Shader (p. 1712)</td>
<td>Use to render trees, bushes, grass, and other vegetation, as well as imparting various bending motion effects.</td>
</tr>
<tr>
<td>VolumeObject Shader (p. 1714)</td>
<td>Use to render various volumetric objects such as clouds, fog, and smoke, and to impart realistic shading and self-shadowing effects.</td>
</tr>
<tr>
<td>Water Shader (p. 1715)</td>
<td>Use to render the ocean exclusively, and to impart various reflection, ripple, and foam effects.</td>
</tr>
<tr>
<td>Waterfall Shader (p. 1717)</td>
<td>Use to render waterfalls exclusively, and provides layering and tiling, as well as motion effects.</td>
</tr>
<tr>
<td>WaterVolume Shader (p. 1719)</td>
<td>Use to render volumetric bodies of water including lakes, pools, and rivers, and to impart various reflection, ripple, and foam effects.</td>
</tr>
</tbody>
</table>

**Common.Cloud Shader**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Common.Cloud shader is used exclusively for common clouds (p. 830). The shader uses per-vertex gradient lighting and takes the sun, cloud, and viewer positions into account. Gradient lighting interpolates between the bright color, which is calculated from the HDR Sun color multiplier, and the
dark color, which is calculated from the HDR Sky color multiplier. In addition, rim lighting is also applied on a per-pixel basis to capture the effects of light scattering when looking at clouds that are lit by the sun from behind.

Common clouds use soft clipping to gradually fade in and out at the near and far clipping plane. This prevents rendering artifacts in the far distance and flickering due to cloud particles entering and leaving the view cone near the camera during a flythrough. Additionally, clouds blend softly against opaque scene geometry.

The default **Diffuse** texture map for this shader is `cumulus_01.dds`, which contains images of cloud particles.
Shader Parameters

CloudAngularAtten

Defines the angular attenuation factor for rim lighting. The smaller the value, the more widespread the rim lighting effect for clouds (partially) covering the sun becomes from the viewer's point of view.

Valid values: 1 to 100

Default value: 30

Example

The following examples show a cloud with angular attenuation set to 20 and 100.

CloudBacklightingScale

Defines how much to scale rim lighting. Higher values increase the glow of cloud edges.

Valid values: 0 to 5

Default value: 1

Example

The following examples show a cloud with backlighting scale set to 0 and 5.
CloudOutlineSlope

Defines the slope of the ramp function used to blend in rim lighting. Higher values create harder transitions.

Valid values: 0 to 20

Default value: 1

Example

The following examples show a cloud with outline slope set to 0 and 20.

CloudOutlineThreshold

Defines the cloud's opacity threshold value below which the rim lighting effect is applied. Higher thresholds cause the rim lighting to grow inward.
Valid values: 0.0 to 1.0

Default value: 0.4

Example

The following examples show a cloud with outline threshold set to 0.0 and 1.0.

HDRBrightnessAdjust

Controls brightness of clouds in high dynamic range (HDR) image format (relative to low dynamic range (LDR) image format).

Default value: 1

DistanceClouds Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The DistanceClouds shader is a dedicated shader used for 2D clouds that are placed at a far distance.

Shader Parameters

Alpha Multiplier

Alpha multiplier for cloud texture.

This parameter requires that the Advance distance clouds shader generation parameter is enabled.

Default value: 1

AlphaSaturation

Controls the alpha saturation of clouds when blending them with the sky. High values make less opaque parts of the cloud texture fade out more.
You can reuse the same texture for slightly different looking clouds by defining several materials with custom \textit{AlphaSaturation} values.

This parameter does not apply if the \textit{Simple distance clouds} shader generation parameter is enabled.

Default value: 2

\textbf{Attenuation}

Controls how strongly sun light is attenuated when traveling through the distance cloud. Light attenuation is computed per pixel.

Use \textbf{Attenuation} to blend between current sun color and sky color. Use higher attenuation values to accentuate cloud self-shadowing (for example, strong cloud layers).

This parameter applies if no Shader Generation parameter is enabled.

Default value: 0.6

\textbf{Cloud Height}

Sets the height of the cloud layer.

This parameter requires that the \textit{Advanced distance clouds} shader generation parameter is enabled.

Default value: 0.3

\textbf{Density Sky}

Sets the cloud density that is used for sky light scattering.

This parameter requires that the \textit{Advanced distance clouds} shader generation parameter is enabled.

Default value: 4.5

\textbf{Density Sun}

Sets the cloud density that is used for sunlight scattering.

This parameter requires that the \textit{Advanced distance clouds} shader generation parameter is enabled.

Default value: 1.5

\textbf{Exposure}

Sets exposure amount to enable HDR on LDR cloud texture.

This parameter requires that the \textit{Simple distance clouds} shader generation parameter is enabled.

Default value: 1

\textbf{Opacity}

Sets opacity modifier for the cloud.

This parameter requires that the \textit{Simple distance clouds} shader generation parameter is enabled.

Default value: 1

\textbf{SkyColorMultiplier}

A value multiplied to the sky color defined for the current time of day.
The result is used in the pixel shader to blend between sun and sky color using the computed light attenuation value.

This parameter applies if no shader generation parameter is enabled.

Default value: 1.5

**StepSize**

Controls how fast to step through the cloud texture (density) to compute per-pixel light attenuation.

This effect determines the appearance of the gradient. Higher values create smoother and less abrupt gradients, but can also produce unnatural gradient changes over time of day.

This parameter applies if no shader generation parameter is enabled.

Default value: 0.004

**SunColorMultiplier**

A value multiplied by the sun color that is defined for the current time of day. The result is used in the pixel shader to blend between sun and sky color using the computed light attenuation value.

This parameter applies if no shader generation parameter is enabled.

Default value: 4

**Shader Generation Parameters**

**Simple distance clouds**

Enables the use of distance clouds with no volumetric shading computations.

**Advanced distance clouds**

Enables the use of distance clouds with more accurate shading computations.

**Eye Shader**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Eye shader is used to render realistic eyes that take sclera, cornea, iris, and eye moisture properties into account.

**Shader Parameters**

**Cornea Refraction**

Controls and optionally animates pupil size.

Default value: 0.01

**Cornea Smoothness**

Controls the glossiness of corneas reflections.
The default creates smaller and sharper highlights that are more lifelike.

Default value: 1

**Indirect bounce color**

Sets the amount of indirectly bounced color. Has no effect when the Physically Based Shading (PBR) model is used.

Default value: 136,136,136

**Iris Color**

Tweaks the iris color without affecting the eye white.

**Iris Color** can be used for eye variation between characters that use the same texture.

Default value: 187,187,187

**Iris Depth**

Simulates the actual form of the iris, since the in-game mesh has the shape of a sphere.

Default value: 0.005

**Iris Shadowing**

Controls iris self-shadowing, which further simulates the actual form of the iris.

*Note*

This effect is only affected by sunlight and not by other light sources.

Default value: 5

**Iris SSS**

Controls the subsurface scattering (SSS) amount of the iris, which blurs the shadows. Higher values blur the shading more.

Default value: 0.6

**Sclera SSS**

Controls the subsurface scattering (SSS) amount of the eye whites, which blurs the shadows. Higher values blur the shading more.

Default value: 0.4

**Depth bias scale**

Sets the depth bias of the overlay mesh to avoid clipping with the eyes.

This parameter requires that the *Specular overlay* shader generation parameter is enabled.

Default value:

**Diffuse occlusion strength**

Controls the strength of the occlusion effect on the eyes.

This parameter requires that the *Ambient occlusion overlay* shader generation parameter is enabled.

Default value: 1
**Specular occlusion strength**

Controls the strength of the occlusion effect on the eyes' specular highlights.

This parameter requires that the Ambient occlusion overlay shader generation parameter is enabled.

Default value: 1

**Shader Generation Parameters**

**Environment map**

Enables environment map as a separate texture.

If the blending cube map feature isn't used, Environment map must be enabled and nearest_cubemap must be assigned for the texture's environment.

**Ambient occlusion overlay**

Enables ambient occlusion overlay rendering.

Must be enabled to use the occlusion mesh that overlays the eye. This mesh gives the eyes a more natural shadowing and integrates them with the head.

**Specular overlay**

Enables the eye water mesh.

---

**Fur Shader**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard's Fur shader renders fur in real time. You can apply this shader to any mesh and adjust its properties to customize the fur's appearance and behavior.

**Topics**

- Fur Combing (p. 1664)
- Fur Features (p. 1665)
- Material Settings (p. 1674)
- Fur Console Variables (p. 1677)
- Maya – Fur Previsualization (p. 1679)
- 3DSMax – Fur Previsualization (p. 1685)

To use the fur shader, create a new material and, from the Shader drop-down, select Fur. Apply this new material to your mesh object.
You set a Fur Heightmap (p. 1674) to define where fur appears, how it clumps together, and to provide non-uniformity of strands across a mesh. A fur heightmap is a grayscale texture, as shown in the following image.

![Fur Heightmap Image]

Brighter values at a UV result in longer fur at that point. Under Shader Params, adjust Fur Length (cm) to modify the length of the fur. The following image shows objects with different fur lengths.
Fur Combing

You can apply fur to any mesh. But by authoring your meshes specifically for fur, you can achieve a more controlled look.
To use fur combing, in Shader Generation Params, enable Fur Color Data. With this parameter enabled, the engine uses the mesh's vertex color to control the fur direction and length. The RGB channels correspond to an XYZ combing direction, and the alpha channel scales the fur length.

For example, a material might have a vertex color of $1, 0.5, 0.5, 0.25$ (X, Y, Z, alpha). These values specify that the fur combs in the positive X direction. The alpha channel, set at $0.25$, defines the fur length at that vertex as one fourth of the value specified in the material's Fur Length (cm).

Lumberyard provides shaders for Maya (p. 1679) and 3DSMax (p. 1685). These shaders simplify the process of specifying the vertex color data on your mesh.

For information on how to use these shaders, see the following topics:

- Maya – Fur Previsualization (p. 1679)
- 3DSMax – Fur Previsualization (p. 1685)

Fur Features

Using various Fur Material Settings (p. 1674) and Fur Console Variables (p. 1677), the fur shader can give your furry entities realistic highlights, dynamic shadows, responsive movement, and much more.

Topics

- Anisotropic Specular Highlights (p. 1665)
- Subsurface Scattering (p. 1667)
- Simulated Self-Shadowing (p. 1668)
- Fur in Shadows (p. 1669)
- Bending (p. 1670)
- Motion Bending (p. 1673)
- Level of Detail (p. 1674)

Anisotropic Specular Highlights

Anisotropic specular highlights are supported when a mesh provides fur combing (p. 1664) data. This feature creates highlights that appear "banded" in the direction perpendicular to the fur strands. Notice the banded highlight from the green light on the bear's fur.
Anisotropic highlights are not applied to areas where fur length is set to 0 in the mesh vertex color. For example, non-furry areas such as a nose, mouth, or claws will display isotropic, or hotspot, highlights rather than banded, or anisotropic highlights.
Subsurface Scattering

The fur shader simulates subsurface scattering around silhouette edges of the fur, where less inscattering and absorption occurs. This results in a glow effect when the fur is backlit. You can tune this feature by adjusting the material parameters **SSS Strength** and **SSS Falloff**.
Simulated Self-Shadowing

To simulate the self-shadowing effect of fur, you can set a blend color or blend layer. To do this, you can enable **Fur Blendlayer** or **Fur Blend color**. If both are enabled, only **Fur Blendlayer** is used. You then set the color using the **Fur Self Shadowing Color** property. This value is interpolated with the default diffuse texture, where self-shadowing is stronger toward the mesh surface.

The following image shows a furry sphere with no self-shadowing (1), the same sphere with fur blend color (2), and again with fur blendlayer (3).
You can adjust the interpolation speed by modifying the material parameter **Fur Self Shadowing Bias**.

**Fur in Shadows**

When fur casts a shadow, it adds definition to the silhouette. If fur is not set to cast a shadow, then you would see a smooth edge of the structure that the fur covers. In the following image, you can see the shadow of the bear's fur on the wall.
Fur can contribute to shadow passes, including bending caused by wind. Fur is alpha-tested in shadow passes.

You can adjust this by modifying the material parameter **Fur Fins Alpha Test**.

You can also turn on fur in shadows by toggling the console variable `r_furfinshadowpass`.

**Bending**

In addition to fur combing by vertex normal, Lumberyard provides three additional types of bending: **Wind Bending**, **Gravity Bending**, and **Motion Bending**. You can control the strength of the bending by adjusting the Material Parameter (p. 1674) **Stiffness**. This parameter specifies the amount to which the fur is resistant to change.

**Wind Bending**

Enable the Shader Generation Parameter (p. 1674) **Wind bending** to see the effects of wind on fur's movement. Adjust the shader parameter (p. 1674) **WindFrequency**, **WindPhase**, and **WindScale** to control how the wind affects the fur. The amount of wind applied to the fur is also dependent on the local wind at the entity position as measured by both global wind and wind areas.
Gravity Bending

Gravity bending is controlled by adjusting the shader parameter (p. 1674) **Fur Maximum Gravity**. Gravity is applied most where the fur sprouts away from the world's surface. It is also applied more strongly to the ends of the fur pieces.

For example, the following image shows a fur ball with a very low gravity applied. The second image shows the same fur ball with a higher maximum gravity applied.
Motion Bending

Motion bending occurs when a furry entity's transform changes, such as its position, rotation, or scale. The fur appears to lag behind the motion slightly, and catches up over a short time. You can tune this globally using the console variable (p. 1677) `r_FurMovementBendingBias`. 
Level of Detail

Fur rendering level of detail uses many of the same mechanics as mesh level of detail.

For fur rendering, the number of shell passes drawn decreases over a distance. You can define the maximum number of passes using the console variable (p. 1677) `r_FurShellPassCount`.

The starting distance for beginning to decrease shell passes is mesh dependent. This distance is determined by the mesh's parameters View distance multiplier and LOD distance ratio. You can find these parameters for the mesh (p. 684) or skinned mesh (p. 684) in the Entity Inspector (p. 475)'s Options section. The ending distance, after which no fur shells are drawn, is determined by the console variable (p. 1677) `r_FurMaxViewDist`.

Material Settings

Using fur material parameters, you can fine-tune the look of the fur.

Fur rendering supports the following textures:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffuse</td>
<td>The texture used for the mesh.</td>
</tr>
</tbody>
</table>
### Shader Reference

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Map</td>
<td>Adjustments for per-vertex normals. Note that a busy normal map may reduce the appearance of anisotropic specular highlights, as the normals do not align across the mesh as well.</td>
</tr>
<tr>
<td>Specular</td>
<td>Specularity of the mesh.</td>
</tr>
<tr>
<td>Fur Heightmap</td>
<td>A grayscale texture that specifies how fur fades out at different frequencies over the mesh. Without this texture, fur appears blurry and without definition to strands.</td>
</tr>
</tbody>
</table>

#### Default fur Shader Params

Default fur **Shader Params** are as follows:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fur Fins Alpha Test</td>
<td>[0.0, 1.0]</td>
<td>Alpha test value used for fin and shadow passes, if enabled.</td>
</tr>
<tr>
<td>Fur Length (cm)</td>
<td>[0.0, 100.0]</td>
<td>Maximum length of fur, in centimeters.</td>
</tr>
<tr>
<td>Fur Maximum Gravity</td>
<td>[0.0, 5.0]</td>
<td>Maximum amount of gravity to apply to fur.</td>
</tr>
<tr>
<td>Fur Stiffness</td>
<td>[0.0, 1.0]</td>
<td>Rigidity of the fur. Higher values cause less bending to occur.</td>
</tr>
<tr>
<td>Indirect bounce color</td>
<td>Color</td>
<td>Tint for bounce light from global illumination.</td>
</tr>
<tr>
<td>SSS Falloff</td>
<td>[0.0, 5.0]</td>
<td>Falloff speed for subsurface scattering influence.</td>
</tr>
<tr>
<td>SSS Strength</td>
<td>[0.0, 5.0]</td>
<td>Initial strength of subsurface scattering influence.</td>
</tr>
<tr>
<td>Threshold for writing depth</td>
<td>[0.0, 1.0]</td>
<td>For alpha blended fur, the alpha threshold for writing depth. Using this, alpha blended fur</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Value Range</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>works well with post effects such as depth of field.</td>
</tr>
</tbody>
</table>

When you enable **Fur Color Data** under **Shader Generation Params**, the following additional parameters appear:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fur Base Intrusion</td>
<td>[0.0, 1.0]</td>
<td>Percentage of scaled fur length per vertex to move the base mesh in the negative normal direction. With this parameter, a mesh can provide a similar silhouette with or without fur.</td>
</tr>
<tr>
<td>Fur Combing Bias</td>
<td>[0.0, 5.0]</td>
<td>Defines how quickly fur combs from the normal direction to the per-vertex combing direction.</td>
</tr>
<tr>
<td>Wind Scale</td>
<td>[0.0, 1.0]</td>
<td>Scale of local wind speed to apply to the fur.</td>
</tr>
</tbody>
</table>

When you enable **Wind bending** under **Shader Generation Param**, the following additional parameters appear in **Shader Params**:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wind Frequency</td>
<td>[0.0, 10.0]</td>
<td>Speed at which the wind moves through a set pattern of amplitudes.</td>
</tr>
<tr>
<td>Wind Phase</td>
<td>[0.0, 10.0]</td>
<td>Phase offset for wind applied to this material.</td>
</tr>
<tr>
<td>Wind Scale</td>
<td>[0.0, 1.0]</td>
<td>Scale of local wind speed to apply to the fur.</td>
</tr>
</tbody>
</table>

When you enable **Fur Blendlayer** or **Fur Blend color** under **Shader Generation Params**, the following additional parameters appear in **Shader Params**:

**Note**

If both **Fur Blendlayer** and **Fur Blend color** are enabled, Lumberyard uses only **Fur Blendlayer**.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fur Self Shadowing Bias</td>
<td>[0.0, 5.0]</td>
<td>Defines how quickly the blendlayer or blend color fades to the primary diffuse texture color.</td>
</tr>
<tr>
<td>Fur Self Shadowing Color</td>
<td>Color</td>
<td>With <strong>Fur Blend color</strong>, the color to apply at the base of the fur.</td>
</tr>
</tbody>
</table>
### Shader Reference

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fur Self Shadowing Map</td>
<td>Texture</td>
<td>With <strong>Fur Blendlayer</strong>, the texture to apply at the base of the fur. Uses the same UV set as the diffuse texture.</td>
</tr>
</tbody>
</table>

When you enable **Scale fur length** under **Shader Generation Params**, the entity's scale is factored into the fur length used by the mesh.

When you enable **Model Space Z Up** under **Shader Generation Params**, fur combing is computed as Z-up instead of Y-up.

**Note**
For materials that use the **Fur** shader, Lumberyard ignores settings for **Opacity** (under **Opacity Settings**) and **Emissive Intensity (kcd/m2)** (under **Lighting Settings**).

### Fur Console Variables

<table>
<thead>
<tr>
<th>Console Variable Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
</table>
| r_Fur                         | [0,2]       | Specifies how fur draws:  
  • 0: Fur does not draw; appears similar to Illum  
  • 1: Transparent alpha blended passes  
  • 2: Opaque alpha tested passes |
| r_FurDebug                   | [0,8]       | Toggles various visual debug information for fur rendering:  
  • 1: Base/tip sample validity (red = base valid; green = tip valid; yellow = both valid)  
  • 2: Base/tip selection (red = base chosen; green = tip chosen)  
  • 3: Show offscreen UVs for base deferred sample (gray = onscreen)  
  • 4: Show offscreen UVs for tip deferred sample (gray = onscreen)  
  • 5: Show final lighting with all base lighting selected  
  • 6: Show final lighting with all tip lighting selected  
  • 7: Visualize fur length scaling  
  • 8: Visualize fur animation bending velocity |
<p>| r_FurDebugOneShell            | [0, ∞ )     | If the specified shell number exists, draws only that shell, |</p>
<table>
<thead>
<tr>
<th>Console Variable Name</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_FurFinPass</td>
<td>[0,1]</td>
<td>Toggles the drawing of an opaque alpha tested silhouette fin pass for fur rendering.</td>
</tr>
<tr>
<td>r_FurFinShadowPass</td>
<td>[0,1]</td>
<td>Toggles the drawing of alpha tested silhouette fins in shadow passes.</td>
</tr>
<tr>
<td>r_FurMaxViewDist</td>
<td>[0.0, ∞)</td>
<td>Specifies the maximum distance for fur shells to be drawn.</td>
</tr>
<tr>
<td>r_FurMovementBendingBias</td>
<td>[0.0, 1.0]</td>
<td>Specifies motion bending bias for fur. A value closer to 0 causes fur motion bending to be more pronounced.</td>
</tr>
<tr>
<td>r_FurShellPassCount</td>
<td>[0, ∞)</td>
<td>Maximum number of fur shell passes to draw for each object.</td>
</tr>
<tr>
<td>r_FurShowBending</td>
<td>[0, 1]</td>
<td>When enabled, draws debug lines visualizing the normals and fur combing vectors on a mesh.</td>
</tr>
</tbody>
</table>
Maya – Fur Previsualization
To visualize fur on a mesh

1. Verify that the Cgfx Shader is enabled.

   In Maya, navigate to Windows, Settings/Preferences, Plug-in Manager.

   Find cgfxShader.mll in the list, and then verify that both Loaded and Auto load are selected.

   If you made any changes, restart Maya.

   ![Plug-in Manager](image)

2. Open the Hypershade editor by navigating to Rendering Editors, Hypershade.
3. Create a new material. To do this, in the **Hypershade** window, navigate to **Create**, **Materials**, **Cgfx Shader**.
4. In the Property Editor, click the folder icon next to the CgFX File text box.

Note
If the Property Editor is not visible, navigate to Windows, Property Editor.

5. Navigate to your Lumberyard install folder, and then select `dev\Tools\Maya\fx\Fur.cgfx`.

6. In the Fur.cgfx Parameters section, specify the following, at a minimum:
   - Enable Fur Combing (disabled/unselected if you haven't prepared your mesh with vertex colors for fur combing)
   - Use Base Fur Diffuse Map (selected/enabled)
7. Select the mesh or sub-mesh on which you want to visualize fur, and then right-click hold on the CgFX shader material in Hypershade. Select Assign Material to Selection.

8. In the main viewport, press 6 to ensure that Shaded Display (with texture maps) is enabled.

9. Adjust fur settings on the material as preferred.

To apply vertex colors on a mesh

1. Enable fur combing on the material. To do this, on the Fur.cgfx Parameters, select Enable Fur Combing.
2. Open the **Paint Vertex Color Tool**. To do this, click the square box next to **Color, Paint Vertex Color Tool**.

3. Since Maya does not permit painting alpha only on a RGBA color set, begin by painting RGBA to specify fur length scaling:
   a. In the **Color** section of the tool, select **RGBA** for the **Channels**.
   b. Click the color swatch next to **Color value**. Specify **0.5, 0.5, 0.5** as the color using the mode **RGB, 0.0 to 1.0**.
   c. Set the alpha value to your desired fur scaling setting.
   d. Set other tool settings as needed, and apply fur scaling to your mesh.

4. Now paint RGB only to apply combing:
   a. In the **Color** section of the tool, select **RGB** as the **Channels**.
   b. Click the color swatch next to **Color value**. Select the preferred direction.
   c. Set other tool settings as needed, and apply fur scaling to your mesh.
3DSMax – Fur Previsualization

To visualize fur on a mesh

1. Ensure that your renderer is set to Legacy Direct3D:
   a. Open the Preference Settings dialog. To do this, navigate to Customize, Preferences.
   b. Select the Viewports tab. Under Display Drivers, view the currently installed driver.
c. If it does not display Direct3D 9.0, click Choose Driver. From the drop-down menu, choose Legacy Direct3D. Click OK.

2. Open the Material Editor. To do this, navigate to Rendering, Material Editor, Compact Material Editor.

3. Select a material that you want to set for fur, and click the button to the right of the submaterial drop-down menu (the button might display “Standard”).
Choose **DirectX Shader**, and choose to discard the old material.

4. Click in filename box. Navigate to your Lumberyard install folder, and select `dev\Tools\maxscript\fx\Fur.fx`.

5. In the **Fur.cgfx Parameters** section, specify the following, at a minimum:
   - **Enable Fur Combing** (disabled/unselected if you haven't prepared your mesh with vertex colors for fur combing)
   - **Use Base Fur Diffuse Map** (unselected/disabled)
   - **Color Texture**
   - **Fur Heightmap**

6. In the viewport, select your mesh or sub-mesh. In the **Material Editor**, click **Assign Material to Selection**.
To apply vertex colors on a mesh

1. With your object selected, go to the Modify panel. From the Modifier List drop-down menu, select VertexPaint.

2. To adjust fur combing, set the Channel to Vertex Color.
   
   To adjust fur length scaling, set the Channel to Vertex Alpha.

3. For fur combing:
   
   a. Click on the color swatch below the eraser button, and select the preferred direction.
   
   b. Use the Paint or Paint All button to apply the combing direction to the mesh.
Note
The default vertex color for meshes is white, so you may want to begin by using Painting All with RGB(128, 128, 128). This way, you can work from a default “no combing” state.

4. For fur length scaling:
   a. Click on the color swatch below the eraser button, and set the preferred length scale in the red channel.
   b. Use the Paint or Paint All button to apply the length scaling to the mesh.

GeometryBeam Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the GeometryBeam shader to create volumetric light beams that feature dust and turbulence effects.

Shader Parameters

Ambience strength
Controls the general strength of the beam effect.
Default value: 0.12

Base UV scale
Controls the scale or tiling of the object’s base UV mapping.
Default value: 1

Brightness
Controls the overall brightness of the beam effect.
Default value: 1

Dust anim speed
Controls the animation speed for the dust turbulence effect, as defined by the Specular texture map.
This parameter requires that the Dust Turbulence shader generation parameter is enabled.
Default value: 1

Dust UV rotation
Changes the rotation of the dust turbulence effect, as defined by the Specular texture map.
This parameter requires that the Dust Turbulence shader generation parameter is enabled.
Default value: 0

Dust UV scale
Sets the scale or tiling of the UV mapping for the dust turbulence effect, as defined by the Specular texture map.
This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

**Default value**: 0.6

**End color**

Sets the end color for gradient along the U axis.

**Default value**: 255,255,255

**Soft intersection factor**

Controls softness of surface interaction with other opaque scene geometry.

**Default value**: 1

**Start color**

Sets the start color for gradient along the U axis.

**Default value**: 255,255,255

**Turbulence tiling**

Multiplies turbulence, as defined by the **Bumpmap** texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

**Default value**: 1

**Turbulence visibility**

Controls the visibility level of turbulence, as defined by the bump map texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

**Default value**: 0.55

**UV vignetting**

Applies a vignetting effect to the edges of the UV map.

This parameter requires that the **UV Vignetting** shader generation parameter is enabled.

**Default value**: 4

**Vertex alpha fading**

If you use vertex alpha to fade out the edges, use this slider to control the interpolation curve.

**Default value**: 0.55

**View dependency factor**

Determines how beams blend in and out depending on the camera-facing angle.

**The higher the value, the longer the beam is visible even when at a nearly 90° angle to camera. Smaller values cause the beam to begin to vanish.**

**Default value**: 2

**Volumetric scale**

Controls the volumetric features when shadow receiving is enabled. This also has the effect of changing the soft shadow radius.

This parameter requires that the **Receive Shadows** shader generation parameter is enabled.
Default value: 0.7

**Shader Generation Parameters**

**Dust Turbulence**

Enables dust and turbulence overlay. Specular and Bumpmap texture map slots also become available under Texture Maps to fine-tune appearance.

**Receive Shadows**

Enables sun shadows to be cast on the light beams, creating volumetric shafts.

You can use this parameter for an interesting effect, but it might affect your game's performance.

**UV Vignetting**

Enables vignettes in UV space.

---

**Glass Shader**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Glass shader renders windows and other glass objects, imparting refractive, tint, fog, and cracking effects for both breakable and non-breakable glass objects. Use the Illum Shader (p. 1697) instead if you require non-refractive effects for non-breakable glass objects.

Here are a few things to keep in mind when using the Glass shader:

- Ambient diffuse lighting from cube maps isn't taken into account.
- The shader uses the sky color exclusively for all ambient lighting.
- Except for the sun, all deferred lights don't affect transparent glass objects.
- The shader can't receive sun shadows.

**Shader Parameters**

**Back light scale**

Controls the amount of light that gets through the glass.

Default value: 0.5

**Blur Amount**

Controls the amount of blur.

This parameter requires that the Blur refraction – PC Only shader generation parameter is enabled.

Default value: 0.5

**Bump Map Tiling**

Adjusts tiling of the bump map independently from diffuse.

Default value: 1
**Bump Scale**
Sets the reflection and refraction bump scale.

Default value: 0.005

**Cloudiness Masks Blur**
Applies blur to just cloudy areas.

This parameter requires that the Tint map – Tint/Gloss/Spec shader generation parameter is enabled.

Default value: 0

**Cloudiness Masks Gloss**
Makes cloudy areas less glossy.

This parameter requires that the Tint map – Tint/Gloss/Spec shader generation parameter is enabled.

Default value: 0.5

**Fog color**
Sets fog color.

This parameter requires that the Depth Fog shader generation parameter is enabled.

Default value: 255,255,255

**Fog cutoff end depth**
Sets the distance, in meters, after which fog doesn’t get any stronger.

This parameter requires that the Depth Fog shader generation parameter is enabled.

Default value: 20

**Fog density**
Sets fog density.

This parameter requires that the Depth Fog shader generation parameter is enabled.

Default value: 1

**Indirect bounce color**
Sets the amount of indirectly bounced color.

Not used if the Depth Fog shader generation parameter is enabled.

Default value: 136,136,136

**Tint Cloudiness**
Adjusts the cloudiness of tinted areas.

Default value: 0

**Tint Color**
Applies a tint color to the glass.
Default value: 255,255,255

**Shader Generation Parameters**

**Use Diffuse map**

Enables diffuse map for dirt, and so on. Requires alpha channel.

**Environment map**

Enables environment map as a separate texture.

**Tint map – Tint/Gloss/Spec**

Enables the RGB spec map to control tinting in red channel, cloudiness in green channel, and specular in blue channel.

**Use Tint Color Map**

Enables the Tint Color map. Used for multicolored glass, which goes in the custom Tint Color map slot.

**Blur refraction – PC Only**

Enables the blurring of objects seen through the glass.

**Depth Fog**

Enables depth fog behind the glass surface.

**Disable Lights**

Disables the reflection of lights.

**Hair Shader**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Hair shader is a dedicated shader for rendering hair and fur, imparting different color, stranding, and animation effects. Hair rendering is a relatively difficult task to achieve in real-time with high-quality results due to the very fine geometry and specific lighting behavior. Depending on the hairstyle, either a simple scalp plane or a more complex shape that defines the volume of the hairstyle is needed. In some cases, breaking up a hairstyle into multiple large patches makes more sense.

**Shader Parameters**

**Alpha Blend Multiplier**

Multiplies the alpha map with the result that grayscale values are increased. Useful for the Thin Hair shader generation parameter.

Default value: 1

**Diffuse Wrap**

Allows light to pass through the hair, thus illuminating a wider area.

A tightly woven braid would have a lower Diffuse Wrap value (the hair being very dense), whereas sparse, loose hair would have a high Diffuse Wrap value.
Default value: 0.5

**Indirect bounce color**

Sets the amount of indirectly bounced color.

Default value: 136,136,136

**Secondary Color**

Sets color and intensity of the secondary specular highlight.

Primary highlight color depends on the diffuse color, whereas the secondary highlight usually has a more neutral color.

Default value: 217,217,217

**Secondary Shift**

Allows the secondary highlight to be shifted over the surface of the hair mesh. Make sure it works with the primary highlight, the position of which can't be shifted.

Default value: 0.1

**Secondary Width**

Sets the width of the secondary specular highlight.

Default value: 1.5

**Shift Variation**

Adds variation to the shift of the secondary highlight.

Default value: 0

**Soft Intersection**

Controls the alpha blending of the hair against skin or scalp.

Default value: 0

**Strand Width**

Controls the width of the view aligned hair strands. The mesh you exported utilizing this feature from DCC tools is rather thin. The value functions as a multiplier relative to the meshes V coordinate (width) in UV space, which can be used to control strand thickness. For example, you might want thinner strands around the border areas.

This parameter requires that the **View aligned strands** shader generation parameter is enabled.

Default value: 0.01

**Thin Hair Threshold**

Determines how alpha blending works for screen space effects such as DOF and motion blur. Lower values make the blending harder but can cause artifacts. Higher values soften the blending, but in some cases the hair turns into a blurry mess.

For most gameplay situations, the rather low default value works fine, but in cinematics, manual tweaking might be needed. The value must then be animated throughout the scene.

This parameter requires that the **Thin Hair** shader generation parameter is enabled.

Default value: 0.05
Wind frequency

Sets the speed at which the vertices are deformed.

This parameter requires that the Wind bending shader generation parameter is enabled.

Default value: 0

Wind phase

Sets hair animation phase and randomizes the deformation.

This parameter requires that the Wind bending shader generation parameter is enabled.

Default value: 1

Wind wave0 amp

Sets the amount or amplitude at which the vertices are deformed.

This parameter requires that the Wind bending shader generation parameter is enabled.

Default value: 0

Wind wave2 amp

Sets the amount or amplitude at which the vertices are deformed on a different curve.

This parameter requires that the Wind bending shader generation parameter is enabled.

Default value: 0

Shader Generation Parameters

Vertex Colors

Enables vertex colors.

View Aligned Strands

Enables the hair strands to self-align to the camera.

Because this is a global setting for the material, using view-aligned strands requires an extra draw call. For more information, see the Strand Width shader parameter.

Thin Hair

For information, see the Thin Hair Threshold shader parameter.

Ambient Cubemap

Enables the use of the nearest cube map specified in environment map slot for ambient lighting. Leave this enabled.

Enforce Tiled Shading

Forces hair to be fully affected by tile shading. This effect works as an override for the global tiled shading settings.

With tiled shading off, improper lighting of a scene can cause hair to turn very dark.

Use this effect carefully, as tiled shading for hair is generally quite expensive.

Wind bending

Simulates wind effects. If enabled, various frequency, phase, and amplitude wind options appear under Shader Parameters.
HumanSkin Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The HumanSkin shader is used to render skin and its various physical properties including color, oiliness, pores, stubble, and wrinkles.

Shader Parameters

**Detail bump scale**

Controls the strength of the detail normal map.

This parameter requires that the Detail normal map shader generation parameter is enabled.

Default value: 0

**Displacement bias**

For information, see Tessellation and Displacement.

This parameter requires that the Displacement mapping shader generation parameter is enabled.

Default value: 0.5

**Displacement height scale**

For information, see Tessellation and Displacement.

This parameter requires that the Displacement mapping shader generation parameter is enabled.

Default value: 1

**Indirect bounce color**

Sets the amount of indirectly bounced color.

Default value: 136,136,136

**Melanin**

Controls the amount of pigmentation in the skin.

Default value: 0

**SSS Index**

Changes the index of subsurface scattering (SSS).

Default value: 1.2

**Tessellation face cull**

This parameter requires that the Displacement mapping shader generation parameter is enabled.

Default value: 0.75

**Tessellation factor**

This parameter requires that the Displacement mapping shader generation parameter is enabled.
Default value: 1

**Tessellation factor max**

This parameter requires that the Displacement mapping shader generation parameter is enabled.

Default value: 32

**Tessellation factor min**

This parameter requires that the Displacement mapping shader generation parameter is enabled.

Default value: 1

**Translucency Multiplier**

Controls strength of the SSS feature.

Default value: 0

**Wrinkles blend**

Controls strength of the wrinkle map.

This parameter requires that the Wrinkle blending shader generation parameter is enabled.

Default value: 1.0

---

**Shader Generation Parameters**

**Decal map**

Enables the use of a decal map, which is blended on top of the diffuse map.

**Detail normal map**

Enables the use of a tiled detailed map for pores and tiny details (_ddn).

**Displacement mapping**

Enables the use of displacement mapping, which requires a height map (_displ).

**Phong tessellation**

Enables the use of rough approximation of smooth surface subdivision.

**PN triangles tessellation**

Enables the use of rough approximation of smooth surface subdivision.

**Subsurface Scattering Mask**

Enables the use of diffuse map alpha as an SSS amount multiplier.

**Wrinkle blending**

Enables the use of subsurface map alpha for wrinkle blending.

---

**Illum Shader**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The Illum shader is the most commonly used shader. You can use this shader to create a variety of effects.

**Shader Parameters**

**Blend Factor**
- Controls the visibility of the blended layer.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.
- Default value: 8

**Blend Falloff**
- Controls blending falloff.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.
- Default value: 32

**Blend Layer 2 Tiling**
- Controls tiling of the second blend layer.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.
- Default value: 1

**Blend Layer 2-Diffuse(Tint)**
- Controls the diffuse (tint) of the second blend layer.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.

**Blend Layer 2-Smoothness**
- Controls the smoothness of the second blend layer.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.
- Default value: 10

**Blend Layer 2-Specular**
- Controls specular intensity of the second blend layer.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.
- Default value: 132, 132, 132

**Blend Mask Tiling**
- Controls tiling of the blend mask.
- To use this parameter, you must enable the **Blendlayer** shader generation parameter.
- Default value: 1

**Detail bump scale**
- Sets detail bump scale.
- To use this parameter, you must enable the **Detail mapping** shader generation parameter.
- Default value: 0.5
**Detail diffuse scale**
Sets diffuse detail blend scale.
To use this parameter, you must enable the `Detail mapping` shader generation parameter.
Default value: 0.5

**Detail gloss scale**
Sets gloss detail blend scale.
To use this parameter, you must enable the `Detail mapping` shader generation parameter.
Default value: 0.5

**Dissolve Color**
Determines color of the edge if edge thickness is greater than 0
To use this parameter, you must enable the `Dissolve FX` shader generation parameter.
Default value: 255,255,255

**Dissolve Edge Thickness**
Determines the thickness of a border that is the `Dissolve Color` around the edge of the effect.
To use this parameter, you must enable the `Dissolve FX` shader generation parameter.
Default value: 0
Valid values: 0.000 to 1

**Dissolve Noise Map**
A grayscale map that defines the dissolve pattern. Black areas dissolve first; white areas dissolve last.
Appears under `Texture Maps` after you enable `Dissolve FX` shader generation parameter.

**Dissolve Percentage**
Controls the amount that the texture is dissolved.
To use this parameter, you must enable the `Dissolve FX` shader generation parameter.
Default value: 0
Valid values: 0.000 to 1

**Height bias**
Controls the height bias.
To use this parameter, you must enable the `Parallax occlusion mapping` shader generation parameter.
Default value: 0.5

**Indirect bounce color**
Adds an extra color tint to the reflection.
Default value: 136,136,136

**OBM Displacement**
Controls the amount of displacement for offset bump mapping (OBM).
To use this parameter, you must enable the **Offset bump mapping** shader generation parameter.

Default value: 0.01

**POM Displacement**

Controls the amount of displacement for parallax occlusion mapping (POM).

To use this parameter, you must enable the **Parallax occlusion mapping** shader generation parameter.

Default value: 0.01

**Self shadow strength**

Allows movable objects, such as interactive objects or game characters, to cast shadows on themselves and each other. Without self-shadowing, for example, if a character holds their right arm over the left, the right arm does not cast a shadow on the left arm.

To use this parameter, you must enable the **Parallax occlusion mapping** shader generation parameter.

Default value: 3

**Roughness maximum footprint**

Specifies the maximum allowed area of the projected pixel footprint at any point in the scene.

To use this parameter, you must enable the **Specular Antialiasing** shader generation parameter.

Valid value ranges: 0–10

**SAA Roughness Boost**

Specifies the degree to which the effect should be applied.

To use this parameter, you must enable the **Specular Antialiasing** shader generation parameter.

Valid value ranges: 0–10

**SSS Index**

Controls subsurface scattering profile (SSS) and the amount.

Valid value ranges: 0.01–0.99 for marble; 1.00–1.99 for skin.

Default value: 1.2

---

**Shader Generation Parameters**

**Detail mapping**

Enables detail mapping. This option requires a **Detail** map, which you can select under the **Texture Maps** heading.

**Use UV set 2 for detail map**

Enables a second UV channel, if available, for detail map.

**Offset bump mapping**

Enables offset bump mapping. This option requires a heightmap, which you can select for the **Normal Map** option under the **Texture Maps** heading.
Dissolve FX

Enables dissolve effect. Selecting this parameter enables the Dissolve Noise Map under Texture Maps, and three Shader Parameters: Dissolve Color, Dissolve Edge Thickness, and Dissolve Percentage.

These three parameters work in tandem to determine the dissolve effect.

For example, assume your Dissolve Color is white, Dissolve Edge Thickness is 0.1, and Dissolve Percentage is 0.5. In that case, any area on the surface of the Dissolve Noise Map that is less than 0.5 (the value set for Dissolve Percentage) is completely transparent. Areas from 0.5 to 0.6, which is a difference of 0.1 (the value set for Dissolve Edge Thickness) are white (the value set for Dissolve Color). Areas from 0.6 to 1.0 are unchanged and appear with the original material.

Vertex Colors

Allows the use of fake ambient occlusion. Also adds more depth and contrast to the model.

You must add vertex colors to the geometry in the DCC tool.

Decal

Applies a decal appearance when enabled for a material. Decal planes are normally placed close to other geometry.

Use this parameter to prevent flickering and z-fighting (p. 3286) when faces are close to each other.

Parallax occlusion mapping (p. 1768)

Enables parallax occlusion mapping. This option requires a heightmap, which you can select for the Normal Map option under the Texture Maps heading.

Displacement mapping

Enables displacement mapping. This option requires a heightmap, which you can select for the Normal Map option under the Texture Maps heading.

Phong tessellation

Enables the rough approximation of smooth surface subdivision.

PN triangles tessellation

Enables the rough approximation of smooth surface subdivision.

Blendlayer

Enables the blending of the normal-mapped diffuse layer on top of the base material.

Use UV set 2 for blendlayer (p. 1763) maps

Enables a second UV channel, if available, for blend layer texture maps and blend mask.

Use UV set 2 for emittance map

Enables a second UV channel, if available, for emittance map.

DetailMap mask in Diffuse alpha

Enables diffuse map alpha for masking detail maps. With this option you can use the alpha channel in the RGBA texture map to mask the decal.

Silhouette POM (p. 1767)

Enables parallax occlusion mapping with silhouettes. This option requires a heightmap, which you can select for the Normal Map option under the Texture Maps heading.

Depth Fixup

Enables write depth for depth of field and postprocessing.
Specular Anti-aliasing

Enables specular anti-aliasing. Enable this feature for highly glossy highlights on curved surfaces, which can be challenging to render without specular shimmering and crawling. This anti-aliasing feature estimates the subpixel region of the scene over which the specular normal distribution function should be filtered. The computation of the region is based on the projected size of each pixel. The normal distribution function (NDF) used in the specular bidirectional reflection distribution function (BRDF) is then filtered over this region, minimizing shimmering, crawling, and other specular aliasing artifacts. This method has almost no impact to GPU performance, is temporally stable, and compatible with Lumberyard's deferred shading system. Normal maps and normal map filtering techniques are compatible with this feature.

Occlusion Map

Enables support for a single channel occlusion map. If provided, this map decreases the amount of influence that indirect light has on a surface. When calculating the occlusion for a given pixel, if the occlusion map occludes more than the dynamically calculated occlusion, then that occlusion map is used instead. Otherwise it is ignored.

Dynamic Lighting for Transparency

Transparent materials (defined as having an alpha less than 100) that use this option are rendered using a full lighting pass. The lighting pass renders the object using a forward pass, forcing the object to be fully lit in any type of light. This parameter creates a better visual effect, but is also more GPU demanding.

LightBeam Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The LightBeam shader creates various fog-like volumetric and atmospheric effects for light beams. The shader can only be applied to light entities. To use the LightBeam shader, create a Projector Light component and assign the shader to the material slot.

Shader Parameters

Fade Distance

Defines the distance at which the effect should fade in/out.

This parameter requires that the Use Falloff shader generation parameter is enabled.

Default value: 200

Fade Scale

Scales how much the fading effect occurs at defined distance.

This parameter requires that the Use Falloff shader generation parameter is enabled.

Default value: 100

Global Density

Controls how dense or thick the fog effect is.

Default value: 1
Jitter Scale

Controls shadow jitter amount. Use to soften shadow artifacts at the cost of shadow accuracy.

Default value: 10

Noise Contrast

Defines the contrast level of the noise effect.

This parameter requires that the Noise map shader generation parameter be enabled.

Default value: 1

Noise Coord Scale

Scales noise. Applies to shadow and projector UVs.

This parameter requires that the Noise map shader generation parameter be enabled.

Default value: 0.005

Noise Dir X

Defines noise travel along the x-axis.

This parameter requires that the Noise map shader generation parameter be enabled.

Default value: 1

Noise Dir Y

Defines noise travel along the y-axis.

This parameter requires that the Noise map shader generation parameter be enabled.

Default value: 0

Noise Dir Z

Defines noise travel along the z-axis.

This parameter requires that the Noise map shader generation parameter be enabled.

Default value: 0

Noise Speed

Controls the speed at which noise travels.

This parameter requires that the Noise map shader generation parameter be enabled.

Default value: 5

Shader Generation Parameters

Noise map

Enables the use of a 3D, procedurally-generated noise map.

Use Falloff

Activates the Fade-type shader parameters to tweak visual fall-off settings.

Extra Sampling

Reduces aliasing for slightly more expensive rendering.
ParticleImposter Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The ParticleImposter shader is used to create particle effects that are not affected by light and hence do not cast shadows or cause reflections.

Particles Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Particles shader is used to render particle effects for fire, smoke, lightning, sparks, and fog that are affected by light, and as such cast shadows and cause reflections.

Shader Parameters

Color lookup amplitude

Sets the color lookup brightness and multiplier.

This parameter requires that the Color lookup shader generation parameter is enabled.

Default value: 1

Color lookup color phase

Sets the per-color phase to be used.

This parameter requires that the Color lookup shader generation parameter is enabled.

Default value: 1

Global Illumination Amount

Sets the amount of global illumination.

Default value: 1

Perturbation amount

Controls the amount of deformation that is used.

This parameter requires that the Screen space deformation shader generation parameter is enabled.

Default value: 0.01

Perturbation anim speed

Controls animation translation speed and frequency that is applied to the deformation map.

This parameter requires that the Screen space deformation shader generation parameter is enabled.

Default value: 0.05
Perturbation tiling

Controls the tiling amount of deformation.

This parameter requires that the Screen space deformation shader generation parameter is enabled.

Default value: 0.5

Deform amount

Controls deformation multiplier.

This parameter requires that the Deformation shader generation parameter is enabled.

Default value: 0

Deform anim speed

Controls deformation animation translation speed and frequency.

This parameter requires that the Deformation shader generation parameter is enabled.

Default value: 0

Deform tiling

Controls deformation tiling.

This parameter requires that the Deformation shader generation parameter is enabled.

Default value: 0.1

Refraction Bump Scale

Sets the refraction bump scale.

This parameter requires that the Refraction shader generation parameter is enabled.

Valid value range: 0 - 2.0

Default value: 0.1

Soft particles scale

Controls soft particle intersection softness for sharper or softer intersections.

Default value: 1

Threshold for writing depth

Sets the threshold for writing depth.

This parameter requires that the Depth Fixup shader generation parameter is enabled.

Default value: 0.05

Shader Generation Parameters

Refraction

Enables the use of a bump-map texture as the displacement for refraction.

Refraction Tinting

Enables the use of a color texture to tint refraction.
Screen space deformation

When enabled, the Refraction Normal texture map slot also becomes available under Texture Maps.

Deformation

When enabled, the Deformation Normal texture map slot also becomes available under Texture Maps.

Color lookup

Enables the use of the color lookup map for applying color lookup. When enabled, the Color Lookup Map texture map slot also becomes available under Texture Maps.

Specular Lighting

Enables the calculation of specular lighting in addition to diffuse lighting.

Depth Fixup

Enables writing depth for depth of field and post processing.

Scopes Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Scopes shader is used to render various optical effects for binoculars, telescopes, and weapon sight scopes.

Shader Parameters

Fake glow amount

Sets the amount of fake glow.

This parameter requires that the Reflex sight new shader generation parameter is enabled.

Default value: 0.25

Fresnel Bias

Sets the amount of fresnel bias.

This parameter requires that the Scope zoomed refraction shader generation parameter is enabled.

Default value: 1

Fresnel Scale

Sets the fresnel scaling amount.

This parameter requires that the Scope zoomed refraction shader generation parameter is enabled.

Default value: 1

Hologram depth

Sets the depth of the hologram.

This parameter requires that the Use halo sight depth shader generation parameter is enabled.
Default value: 2

**Holographic noise scale**

Sets the holographic noise scale.

This parameter requires that the Reflex sight new shader generation parameter is enabled.

Default value: 0

**Noise bias**

Sets noise bias.

This parameter requires that the Reflex sight new shader generation parameter is enabled.

Default value: 1

**Noise scale**

Sets noise scale.

Default value: 0.75

**Object space UV usage**

Sets the amount of usage of object space.

Default value: 0

**Refraction Bump Scale**

Sets the amount of scaling for refraction bumpiness.

Default value: 0

**Scope color multiplier**

Sets the scope color multiplier.

Default value: 160

**Scope scale**

Sets scope scale.

Default value: 4

**Shader Generation Parameters**

**Reflex sight**

Use for reflex-style weapon sights. When enabled, the Diffuse texture map slot under Texture Maps also becomes available.

**Reflex sight new**

Use for the newer version reflex-style weapon sights. When enabled, the Diffuse texture map slot under Texture Maps also becomes available.

**Scope zoomed refraction**

Use to produce light refraction effects for zoomed-in scopes.

**Use halo sight depth**

Used for holographic-style weapon sights with a depth-field modifier.
Thermal vision scope

Use to produce thermal color effects for night-use scopes.

Sky Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
Download O3DE or visit the AWS Game Tech blog to learn more.

The Sky shader is used to render performance-optimized static sky (SkyBox) effects.

Shader Parameters

Indirect bounce color

Adds an extra color tint to the reflection.

Default value: 136,136,136

SSS Index

Subsurface Scattering Index

Default value: 0

SkyHDR Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
Download O3DE or visit the AWS Game Tech blog to learn more.

The SkyHDR shader is used to render realistic dynamic sky effects that change based on the time of day in a level.

Shader Parameters

Indirect bounce color

Adds an extra color tint to the reflection.

Default value: 136,136,136

SSS Index

The Subsurface Scattering Index.

Default value: 0

Shader Generation Parameters

No moon

Removes the moon for the dynamic sky.
No night sky gradient

Removes the entire day night effect gradient for the dynamic sky.

No day sky gradient

Removes the entire day sky effect gradient for the dynamic sky.

TemplBeamProc Shader

The TemplBeamProc shader is used to create inexpensive fog-like light beam effects, enabling control over beam size and blending.

Best Practices

The following are some best practices for using this shader:

- Select the No Shadow property under Advanced.
- Set Opacity to 100%.
- Use a simple grayscale texture with no alpha in the Diffuse texture map slot.
- The shader fades out rendering faces that are at a certain angle to the camera. As such, use different sub-materials for the top plane and the intersecting planes to allow control of the angle of visibility.

Shader Parameters

ColorMultiplier

Increases or decreases brightness and blending.

Default value: 1

EndColor

Sets the end color for the gradient.

Default value: 255,255,255

EndRadius

Sets the radius (in meters) of the effect at the end of the object.

Default value: 2

Length

Adjusts the scaling of the rendered effect.

Default value: 10

OriginalLength

Sets the length scaling factor. If the values of Length and OriginalLength are identical, the object has scale of 100%.

Default value: 10
OriginalWidth

Sets the width scaling factor. If the values of Width and OriginalWidth are identical, the object has scale of 100%.

Default value: 1

Soft intersection factor

Controls softness of surface interaction with other opaque scene geometry.

Default value: 1

StartColor

Sets the start color for the gradient.

Default value: 255,255,255

StartRadius

Sets the radius (in meters) of the effect at the start of the object.

Default value: 1

View dependency factor

Controls the blending in and out depending on the facing angle to the camera.

The higher the value, the longer the effect is visible even when nearly 90° to camera, the smaller the value the earlier the effect starts to vanish.

Default value: 2

Shader Generation Parameters

Noise map

Enables the use of a 3D animated noise map, which enables a nice motion to the beams. However, this motion cannot be controlled by any parameters.

Muzzleflash

Enables use as a muzzle flash effect.

Terrain.Layer Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Terrain.Layer shader is used for painting and blending terrain texture layers in a level. Besides needing a bump map and high-passed diffuse map, the Terrain.Layer shader also requires a height map with either offset bump mapping (OBM) or parallax occlusion mapping (POM) enabled. Blending uses the height map to determine how the materials blend together. For example, if you have pebbles on one material and dirt as another, you may want the pebbles to accurately stand out from the dirt.

Here are a few notes regarding usage of this shader:

- The Detail normals texture is not an external texture, but rather a texture generated by Lumberyard through code.
• The **Decal** parameters don't appear under **Shader Params** unless you put a texture into the **Decal** slot first. The **Decal Bumpmap** slot also appears after this task.
• Flow map textures go in the **Detail** slot.

**Shader Parameters**

**Blend Factor**

Changes the visibility of the blended layer. A height map is required. **OBM** or **OBM** shader generation parameter must be enabled first.

Default value: 0

**Blend Falloff**

Changes the falloff of blending. A height map is required. **OBM** or **OBM** shader generation parameter must be enabled first.

Default value: 1

**Detail bump scale**

**Detail mapping** shader generation parameter must be enabled first.

Default value:

**Detail gloss scale**

**Detail mapping** shader generation parameter must be enabled first.

Default value:

**DetailTextureStrength**

Sets the strength of the diffuse map, which dictates how much detail texture is visible over the layer texture. The higher the value, the more you see only your Diffuse map.

Default value: 1

**Height bias**

**POM** shader generation parameter must be enabled first.

Default value: 0.5

**Indirect bounce color**

Sets the amount of indirectly bounced color

Default value: 136,136,136

Default value:

**OBM Displacement**

**OBM** shader generation parameter must be enabled first.

Default value: 0.01

**POM Displacement**

**POM** shader generation parameter must be enabled first.

Default value: 0.01
Self shadow strength

- **POM** shader generation parameter must be enabled first.
  - Default value: 3

Shader Generation Parameters

**Offset bump mapping (OBM)**

- Uses offset bump mapping. Requires a height map (_displ format).

**Detail mapping**

- Uses detail mapping.

**Parallax occlusion mapping (POM)**

- Uses parallax occlusion mapping. Requires a height map (_displ format).

Vegetation Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Vegetation shader is used to render trees, bushes, grass and other vegetation, as well as imparting various bending motion effects.

See the following guidelines for best results and performance with this shader:

- Use an **AlphaTest** value of 50 for opacity.
- Use a **Diffuse** color value of 128, 128, 128 for lighting.

Shader Parameters

**Bending branch amplitude**

- Defines the movement of blue color in the complex bending setup.
  - Default value: -0.5

**Bending edges amplitude**

- Defines the movement of red color in the complex bending setup.
  - Default value: 0.2

**Blend Factor**

- Changes visibility of blending layer. **Blendlayer** generation parameter must be enabled first.
  - Default value: 0

**Blend Falloff**

- Changes the falloff of blending.
  - Default value: 1
Blend Layer 2 Spec

Changes specular intensity of second blend layer. **Blendlayer** generation parameter must be enabled first.

Default value:

Blend Layer 2 Tiling

Changes tiling of second blend layer. **Blendlayer** generation parameter must be enabled first.

Default value:

Blend Mask Tiling

Changes tiling of blend mask.

Default value: 1

Cap opacity fall off

Controls the fading of alpha test textures when seen at a steep angle (so they look less like a plane). A value of 1 means it's turned off; 0 means it's fully activated.

Default value: 1

Detail bending frequency

Defines the bending speed for complex (wind) bending. Make sure that this value is in the correct proportion to the wind in your level.

Default value: 1

Indirect bounce color

Sets the amount of indirectly bounced color.

Default value: 136, 136, 136

Terrain Color Blend

Controls how much of the terrain color is blended into the diffuse color when up close. **Use Terrain Color** for the selected vegetation object must be enabled first, except when **AutoMerge** is enabled.

Default value: 0

Terrain Color Blend Dist

Controls how much of the terrain color is blended into the diffuse color at a distance. **Use Terrain Color** for the selected vegetation object must be enabled first, except when **AutoMerge** is enabled.

Default value: 0.5

Transmittance Color

Applies color tint for translucency. **Leaves** or **Grass** shader generation parameter must be enabled first.

Default value: 255, 255, 203

Shader Generation Parameters

Leaves

Enables leaf shading and leaves animation. This parameter causes the gaming Lumberyard to use a much more complex (expensive) shading, so activate only for leaves rendering.
Grass

Enables simple and cheap grass rendering. Specular and normal map setting are essentially disabled, so the shading is only diffuse.

Detail bending

Enables detail bending, which simulates wind on vegetation objects. Activate for leaves and grass only. Also, make sure to paint required vertex colors.

Detail mapping

Enables detail mapping.

Blendlayer

Enables normal-mapped diffuse layer blended on top of base material.

Displacement mapping

Enables displacement mapping. Requires a height map (.displ format).

Phong tessellation

Enables rough approximation of smooth surface subdivision.

PN triangles tessellation

Enables rough approximation of smooth surface subdivision.

VolumeObject Shader

Use Amazon Lumberyard's VolumeObject shader to render volumetric clouds with realistic shading and self-shadowing effects.

In addition to the shader parameters, the Time of Day Editor's Cloud Shading parameters (p. 1298) also can affect VolumeObject rendering. To disable this feature, deselect Use TOD Settings in the Shader Generation Parameters (p. 1714).

Shader Parameters

Global Density

Defines how dense the clouds appear.

Default value: 1

Shader Generation Parameters

Soft Intersections

Enhances transparency with opaque scene geometry. Use this parameter sparingly due to increased pixel shading cost.

Back Lighting

Enables back lighting of volume objects. The silhouette slightly glows when viewed against the sun.
Jittering

Enables jittering on volume objects.

Soft Jittering

Softens the jittering effect on volume objects.

Use TOD Settings

Enables time of day (TOD) settings.

Water Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Water shader is a dedicated shader used to render the ocean exclusively, and imparts various reflection, ripple, and foam effects. For lakes, rivers, and other bodies of water, use the VolumeObject Shader (p. 1714) instead.

Shader Parameters

Crest Foam Amount

Sets amount of foam that appears at the crest of a wave. Use for FFT-displaced ocean only on the Very High Spec setting. Foam shader generation parameter must be enabled first.

Default value: 1

Detail Normals scale

Sets normal scale.

Default value: 0.5

Detail Tiling

Sets waves detail bump tiling.

Default value: 2.5

Fake camera speed

Causes the surface of the water to scroll in world-space. This parameter gives the impression that a stationary object in the ocean is actually moving through the ocean. Fake camera movement shader generation parameter must be enabled first.

Default value: 0

Foam Amount

Multiplier for foam. Foam shader generation parameter must be enabled first.

Default value: 1

Foam soft intersection

Very similar to soft intersection, but blends foam on intersection regions. Foam shader generation parameter must be enabled first.

Default value: 0.75
Foam tiling
Sets tiling amount for foam. **Foam** shader generation parameter must be enabled first.
Default value: 12

Fresnel gloss
The gloss of the Fresnel effect.
Default value: 0.9

Gradient scale
Applies a more choppy look to waves.
Default value: 0.1

Height scale
Sets scale for height map, which is used for parallax mapping approximation.
Default value: 0.2

Normals scale
Sets overall scale for normals.
Default value: 1.25

Rain ripples tiling
Sets tiling for rain ripples.
Default value: 1

Reflection bump scale
Reflection map bump scale.
Default value: 0.1

Reflection scale
Sets real-time reflection map multiplier or cube map multiplier for water volumes.
Default value: 1

Ripples normals scale
Sets dynamic ripples normals scale.
Default value: 1

Soft intersection factor
Sets water soft intersection with geometry.
Default value: 1

SSS scale
Sets SSS scale.
Default value: 2

Tiling
Sets waves bump tiling.
Default value: 10
**Watervol flow speed**

Default value:

Sets the flow speed for the water volume flow map. **Water Volume flow** shader generation parameter must be enabled first.

Default value: 10

**Shader Generation Parameters**

**Water Volume flow**

Enables water flow along UVs.

**Water Volume**

Disable this parameter to use the Water shader.

**Sunshine**

Enables sunshine effects on the ocean surface.

**Fake camera movement**

Enables fake camera movement for scenes in the ocean.

**No refraction bump**

Disables refraction bump.

**Foam**

Enables foam on the ocean surface.

**Waterfall Shader**

---

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

---

The Waterfall shader is used for waterfalls exclusively, and provides layering, tiling, and motion effects.

**Shader Parameters**

**Alpha blend multiplier**

Applies a multiplier amount for alpha blending.

Default value: 1

**Foam deform**

Deforms the foam texture with a multiplier, based on the bump map texture. **Foam** shader generation parameter must be enabled first.

Default value: 0.025

**Foam multiplier**

Applies a multiplier amount for foam texture. **Foam** shader generation parameter must be enabled first.
Default value: 1

Fresnel bias

The Fresnel bias.

Default value: 0.25

Layer0 bump scale

Scales the bump map texture for the first layer.

Default value: 2

Layer0 speed

Controls the texture rolling speed for the first layer.

Default value: 1

Layer0 tiling

Sets the texture tiling amount for the first layer.

Default value: 1

Layer1 bump scale

Scales the bump map texture for the second layer.

Default value: 1

Layer1 speed

Controls the texture rolling speed for the second layer.

Default value: 2

Layer1 tiling

Sets the texture tiling amount for the second layer.

Default value: 2

Reflect amount

Controls the reflection amount, which comes from the environment map. Environment map shader generation parameter must be enabled first.

Default value:

Refraction bump scale

Scale the refraction effect inherited by the bump map texture.

Default value: 0.01

Sun multiplier

Applies a multiplier amount for sun shading. Sun shading shader generation parameter must be enabled first.

Default value: 1

Shader Generation Parameters

Environment map

Enables the use of an environment map as a separate texture.
Sun shading

Enables sunlight shading effects.

Foam

Enables foam rendering. Uses diffuse texture.

WaterVolume Shader

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Watervolume shader is used for rendering volumetric bodies of water including lakes, pools, and rivers and imparts various reflection, ripple, and foam effects. For the ocean, use the Water Shader (p. 1715) instead. Here are a few notes regarding usage of this shader:

- The Detail normals texture is not an external texture, but rather a texture generated by Lumberyard through code.
- The Decal parameters don't appear under Shader Params unless you put a texture into the Decal slot first. The Decal Bumpmap slot also appears after this task.
- Flow map textures go in the Detail slot.

Shader Parameters

Detail normals scale

Scales the detail bump normals intensity.

Default value: 0.5

Detail tiling

Sets detail bump tiling.

Default value: 2.5

Env projection scale

Controls the projection scale, or the tiling, of the specified environment map.

Default value: 20

Env reflection amount

Controls the reflection amount of the environment map. Can be offset with Specular Color.

Default value: 1

Flow map scale

Controls the scale, or tiling, of the flow map texture. Enable Water flow map in Shader Generations Params to use this property.

Default value: 0

Flow speed

Specifies the speed of the flow effect. Water flow shader generation parameter must be enabled first.
Default value: 0

**Foam amount**

Controls the amount of foam placed on the water surface. *Foam* shader generation parameter must be enabled first.

Default value: 1

**Foam soft intersection**

Controls how the foam behaves from contact areas. Foam forms around intersecting objects and the terrain after it gets close to the surface. *Foam* shader generation parameter must be enabled first.

Default value: 0.75

**Foam tiling**

Sets the tiling amount of the foam texture. *Foam* shader generation parameter must be enabled first.

Default value: 12

**Normals scale**

Controls the scale of the normals. Don’t confuse this parameter with Detail normals.

Default value: 1.25

**Rain ripples tiling**

Sets the tiling amount for the rain ripples texture.

Default value: 1

**Realtime reflection amount**

Controls the reflection amount for the Realtime Reflection.

Default value: 1

**Soft intersection factor**

Similar to the Foam soft intersection but for the base water surface.

Default value: 1

**Tiling**

Changes the amount of texture map tilling on the water surface.

Default value: 10

**Vertex wave scale**

Sets strength of vertex displaced wave animation.

Default value: 0.125

---

**Shader Generation Parameters**

**Realtime Reflection**

Enables approximate real-time reflections.
Selecting Material Surface Type

Water flow
- Enables water to flow along geometry UVs.

Water flow map
- Enables water flow along a flow map.

Water flow map strength
- Enables additional water flow strength controls, which requires the blue channel for strength.

Sun specular
- Enables water sunshine.

Debug flow map
- Enables visualizing flow map.

Foam
- Enables foam.

Selecting Material Surface Type

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The surface of a material determines the physical effects and how the material reacts to other materials and its environment. For example, a metal surface is hard, doesn't shatter, reacts to bullets by generating spark particles, and has a unique sound when struck. Contrast this with a grass surface, which is soft, responds to wind, generates grass strands and dirt particles when hit, and sounds different than metal.

To select a material surface type
1. In Lumberyard Editor, click Tools, Material Editor.
2. In the left pane, click to select the desired asset.
3. Under Material Settings, make a selection for Surface Type.

Setting Material Opacity

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

An object's opacity refers to its transparency level. Opacity is important when using an alpha channel for transparency.

To set opacity for a material
1. In Lumberyard Editor, choose Tools, Material Editor.
2. In the Material Editor, search for or browse to your asset and then select it.
3. Use the opacity settings to switch between alpha blend and alpha test modes. With the alpha blend mode, you can enable soft and semi-transparent opacity. The alpha test mode is more performant and uses a black and white hard edge from the alpha map.
Under **Opacity Settings**, you can adjust the following parameters:

- **Opacity** – Set the opacity mode and transparency amount of the material. For the alpha blend mode, set the value between 0 to 99. For the alpha test mode, set the value to 100.

  ![Example images for opacity](image)

  - Opacity = 99, AlphaTest = 0
  - Opacity = 99, AlphaTest = 10
  - Opacity = 50, AlphaTest = 0

- **Alpha Test** – Bias the edge of the alpha towards black or white. Set the value below 50 to bias toward the white of the alpha map. Set the value above 50 to bias toward the black of the alpha map.

  ![Example images for alpha test](image)

  - Opacity = 100, AlphaTest = 0
  - Opacity = 100, AlphaTest = 1
  - Opacity = 90, AlphaTest = 50

- **Additive** – Enable to add the material color to the background color. This results in a brighter color for the object.

## Setting Material Lighting and Color Settings

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Material Editor to specify color, specular reflection, and lighting effects such as specularity, glossiness, and glow.
Specify the diffuse, specular, or emissive color values in RGB format, or select the values using the Colors dialog box. Click the color square next to the parameter to open the Colors dialog box. You can then use the color picker to select standard or custom colors for the hue, saturation, and luminance values.

To set lighting and color settings for a material

1. In Lumberyard Editor, click Tools, Material Editor.
2. In the left pane, select the asset that you want to modify.
3. Under Lighting Settings, set values for the following parameters:

### Material Lighting and Color Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffuse Color</td>
<td>The base color of a material.</td>
</tr>
<tr>
<td>Specular Color</td>
<td>The reflective brightness and color of a material when light shines on the object. The greater the value, the shinier the material. Grayscale values affect reflective brightness levels. To tint the specular color, use the color picker to select the desired tint.</td>
</tr>
<tr>
<td>Smoothness</td>
<td>The acuity or sharpness of a specular reflection. For values of 10 or less, there is a scattered reflection. For values greater than 10, there is a sharp reflection. You cannot have glossiness without specular color (reflection), as glossiness determines the sharpness of the reflection.</td>
</tr>
<tr>
<td>Emissive Color</td>
<td>Enable objects to emit light and be visible in the dark. Use this parameter to add brightness to objects. Unlike glow, this parameter does not emit light onto other objects. It does not work with deferred shading.</td>
</tr>
<tr>
<td>Emissive Intensity</td>
<td>Enable objects to glow, and simulate light that emits from extremely bright surfaces. Use this parameter in dark scenes for computer monitors, lamps, fire, neon lights, and similar objects. Unlike emissive color, this parameter emits light onto other objects. You can use a diffuse texture RGB channel to specify glow color, and a diffuse texture alpha channel to specify glow mapping. This allows you to mask the pixels where you want less (or no) glow. You can use glow with the Cloth, HumanSkin, and Illum shaders. To enable or disable glow, use the r_Glow console variable.</td>
</tr>
</tbody>
</table>

### Material ID Mapping in Autodesk 3ds Max

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A mesh (.cgf file) can have different materials assigned to different faces. When you work in Autodesk 3ds Max, make sure you have enough submaterials to cover the number of material IDs assigned to faces on the mesh object. Otherwise the material IDs won’t get exported correctly to Lumberyard.

The following procedure presents an example that uses a multimaterial cube.
To map multi-material IDs in 3ds Max

1. Open 3ds Max. Then create and place a cube in the viewport.

2. Right-click the cube and click **Convert To, Convert to Editable Mesh**. You can now assign different material IDs to the faces.
3. From the 3ds Max top menu, choose Rendering, Material Editor, Compact Material Editor.
4. From the 3ds Max top menu, choose Rendering, Material/Map Browser.
5. In Material/Map Browser, under Materials, expand Standard. Then double-click Multi/Sub-Object. In the 3ds Max Material Editor, under Multi/Sub-Object Basic Parameters, look for a material ID list to fill in. Select the first entry by clicking None in the Sub-Material column. Select Standard under the Standard material rollout.
6. In the 3Ds Max Material Editor, under Shader Basic Parameters, select Crytek Shader.

7. Under Maps, next to Diffuse Color, select None.

8. In Material/Map Browser, under Maps, double-click Bitmap. Then double-click to select the desired image file. Afterward the image file appears in the 3DS Max Material Editor for the Diffuse Color parameter.
9. While still in Material Editor, choose Navigation, Go to Parent. Then repeat to get back to the material ID list.
10. Create a second subshader by repeating steps 5 through 9 for the second entry in the list. Click Set Number, then enter 2 in the Number of Materials pop-up window. The list shows only two submaterials.

11. In Material Editor, under Name, enter a name.

12. With the object selected in the viewport, go to Material Editor and choose Material, Assign to Selection.

13. Click the hammer icon. Under Utilities, select Lumberyard Export, select the object, and then choose Add Selected to place the object in the Geometry Export list.

14. In the 3ds Max panel on the right, under Modifier List, select Editable Mesh, Polygon.

15. In the viewport, select the top face. Then, under Surface Properties, click Set ID and set the value to 2. This makes the top face use the second material in the final material group.

16. Select the other faces and set their Set ID values to 1. The final face coloring should match the one shown in the following image.
17. Select **Export Nodes** to create a `.cgf` file.
18. Click Create Material to open the Lumberyard Material Editor and display a file dialog box.

19. Navigate to the directory where your .cgf files are located. Then enter the same file name that you specified in 3ds Max. This ensures that the .cgf file can automatically find the correct .mtl file when loaded in the Lumberyard Material Editor.
20. In Lumberyard Editor, create a level and open the `.cgf`. The object should have the correct materials mapped onto its faces.

**Working with Textures**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

Textures can be used to provide color, depth, and details to a surface. For example, a repeating brick-and-mortar texture can be used to simulate a brick wall, rather than creating geometry for each individual brick.

A texture is an image file that consists of a number of pixels, called texels, each occupying a coordinate determined by the width and height of the texture. These coordinates are then mapped into values ranging from 0 to 1 along a U (width) and V (height) axis. This process produces a 2D texture map that is stored in a `.DDS` file.

In turn, the process of mapping the UV coordinates of a texture map to the corresponding UV coordinates at the vertices on a 3D object is called UV mapping. This in effect wraps the 2D texture onto the 3D object.

Textures are dictated by, and applied by, the shader that is selected for a material. There can be multiple textures applied by the shader for a material.

Textures used in Lumberyard are usually created with Adobe Photoshop or other DCC tool.

Topics
Texture Map Types

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Source texture files are converted and compiled in .DDS format by the Resource Compiler (RC). If you do not specify presets for the source file, the Resource Compiler will do the following:

- Files with a file suffix of _ddn or _bump will generate an uncompressed RGBA or U8V8 NormalMap .DDS file with height information in the alpha channel.
- Files with a non-white (less than 255) alpha channel will generate a DXT3-compressed .DDS file.
- Files without an alpha channel will generate DXT1 compressed .DDS file.

<table>
<thead>
<tr>
<th>Texture Map</th>
<th>Filename Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffuse map</td>
<td>_diff</td>
<td>Defines the main color for an object.</td>
</tr>
<tr>
<td>Normal map</td>
<td>_ddn</td>
<td>Defines the direction of normals on an object surface. The RGB normal map stores the normal direction for each pixel in the texture.</td>
</tr>
<tr>
<td>Normal with Gloss map</td>
<td>_ddna</td>
<td>Achieves physically-correct results. DDNA textures are standard DDN textures with the gloss map stored inside the Alpha channel.</td>
</tr>
<tr>
<td>Environment map</td>
<td>N/A</td>
<td>Makes an object reflective. The environment map stores the image that is reflected off the object.</td>
</tr>
<tr>
<td>Displacement map</td>
<td>_displ</td>
<td>Gives more depth and definition to an object when used in tessellations, parallax occlusion mapping (POM), and offset bump mapping (OBM).</td>
</tr>
<tr>
<td>Detail map</td>
<td>_detail</td>
<td>Adds more detail to a surface. This texture map works like a second material layer and is not affected by the mapping of the object.</td>
</tr>
<tr>
<td>Emittance Multiplier</td>
<td>N/A</td>
<td>Multiplies the emissive color.</td>
</tr>
<tr>
<td>Blend (layer)</td>
<td>N/A</td>
<td>Blends multiple textures using an adjustable mask texture and a vertex alpha.</td>
</tr>
</tbody>
</table>
Texture Settings Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Texture Settings Editor to edit the texture settings for individual images. This is useful if you are customizing your images for different platforms, such as PC and Android. For example, you can specify compression scheme, mipmap generation parameters, alpha map combinations, and so on for images. When you edit your texture settings for an image, the Texture Settings Editor generates a .imagesettings file to save your settings.

The Texture Settings Editor supports the following image formats:

- .bmp
- .gif
- .jpg
- .jpeg
- .png
- .tga
- .tif
- .tiff

Note

- The Texture Settings Editor requires the Image Processing (p. 1147) gem. By default, this gem is enabled.

The Texture Settings Editor can also load the previous Resource Compiler (RC) generated .exportsettings file. If you make any changes to this file, the Texture Settings Editor deletes this file and generates a new .imagesettings file. If you have Perforce enabled, Perforce tracks the new or modified files.

You can find the default settings for the Texture Settings Editor in the ImageBuilderDefaultPresets.settings file. You can modify the default settings file as needed.

To modify the default settings

2. Navigate to the lumberyard_version/dev/game_project/Config/ImageBuilder/ directory.
3. Paste the copied file and rename it to ImageBuilderPresets.settings.
4. In a text editor, make your changes and save.

Note

This file replaces the previous rc.ini file, which contains the predefined presets to generate textures for your game. To migrate your settings from the rc.ini file, see Migrate RC.ini Settings to the Texture Settings Editor (p. 1746).

Topics

- Using the Texture Settings Editor (p. 1735)
Using the Texture Settings Editor

To use the Texture Settings Editor

1. In Lumberyard Editor, choose **Tools, Asset Browser**.
2. In the **Asset Browser**, navigate and select a source file, such as a `.tif` file.

![Asset Browser Screenshot](image)

You can preview the texture, its file size, dimensions, and select the drop-down menu to preview the texture with the following channels:

- **RGB**
- **Alpha**
- **RGBA**

3. To view more information, expand the arrow icon, and select the texture file that appears below the source file.
See the following information about the texture:

- Compiled texture
- File size
- Resolution
- Format
- Mip count
- Memory size
- Color space
- Image flag

4. To open the Texture Settings Editor, right-click the source file, and choose Edit Texture Settings.

5. In the Texture Settings Editor, you can view the following:
   
   A. **Preview (p. 1737)** – Preview of the processed image with the current settings.
   
   B. **Texture Presets (p. 1738)** – List of presets to choose.
C. **Platform (p. 1739)** – Specify the different resolution adjustments on different platforms.

D. **Mipmap Settings (p. 1743)** – Allows the adjustment of mipmap generation.

6. When finished, click **Apply** to save the current properties. This exports a setting file named `imageName.imagesettings` next to the source file.

**Note**

- If there is an existing legacy `.exportsettings` file associated with the texture file, the legacy file will be automatically deleted.
- Click the ? icon to open the texture documentation.

**Preview**

In the preview panel, you can do the following:

A. Click the drop-down menu to preview the texture in the following channels:

- **RGB** – Displays the RGB color channels (default).
- **R** – Previews the red channel in grayscale.
- **G** – Previews the green channel in grayscale.
- **Alpha** – Previews the alpha channel in grayscale.
- **RGBA** – Previews the alpha blended with the background.
B. **Preview tiled** – Preview the texture in a 2x2 tiled layout. By default, the texture appears in a fixed size.

C. Press and hold the following shortcuts to change the preview for the texture:
   - **Shift** – Changes the preview to RGBA mode.
   - **Alt** – Changes the preview to Alpha mode.
   - **Space** – Opens a pop up window that previews the texture in its actual output size.

D. Click the arrow icons to view the different mipmap previews. Mip 0 is the original output image.

E. Click the refresh icon to refresh the preview. You can click the drop-down menu to specify the following:
   - **Always refresh preview** – Refresh automatically when you make a change.
   - **Press to refresh preview** – Manually refresh the preview.

---

**Texture Presets**

In the **Texture presets**, you can do the following:

- **Active preset** – Click the drop-down menu to view the presets that you can assign to a texture. The texture using the preset values automatically appears in the preview panel. Texture properties are also updated.

  **Note**
  The Texture Settings Editor supports different presets that specify the options that are applied when an image asset is compiled. When you open the Texture Settings Editor, it loads the main settings and presets from the game project folder at the following file:
  `lumberyard_version\dev\game_project\Config\ImageBuilder\ImageBuilderPresets.settings`
  If this file doesn't exist, the Texture Settings Editor loads the default settings from the following file:
  `lumberyard_version\dev\Gems\ImageProcessing\Code\Source\ImageBuilderDefaultPresets.settings`

- Click the reset icon to reset all texture properties to the default values of the current preset.
- Click the info icon to display a pop-up window for the current settings.
• **Active file conventions** – Name convention that the current preset setting supports. For more information, see Texture Map Types (p. 1733).

• **Suppress spec reduction** – By default, when Lumberyard Editor switches from high to low specification, textures are automatically scaled down so that they can load into the graphics card memory. However, you can set this setting, which overwrites all resolution reduction settings, so that the texture keep its original size throughout the different quality settings. It's recommended that you set this setting for textures that have text.

  For more information, see Editor Settings (p. 189).

**Platform**

In the **Platform**, you can specify the following.

---

**Setting**

Lumberyard supports the following platforms:

- PC
- Android
- iOS
Setting

- macOS
**Setting**

*Specifies the resolution for each platform.*

You can specify the following values:

- **0**
  - No reduction.
- **1**
  - Reduces the texture resolution by ½. Reduces the memory consumption to ¼ of the original memory.
- **2**
  - Reduces the texture resolution by 1/4. Reduces the memory consumption to 1/16 of the original memory.
### Setting

- **3**  
  Reduces the texture resolution by 1/8. Reduces the memory consumption to 1/64 of the original memory.

- **4**  
  Reduces the texture resolution by 1/16. Reduces the memory consumption to 1/256 of the original memory.

- **5**  
  Reduces the texture resolution by 1/32. Reduces the memory consumption to 1/1024 of the original memory.
**Setting**

**Size**
Compiled texture resolution calculated according to the **DownRes** value.

**Format**
Compiled texture format on the platform.

### Mipmap Settings

In the **Mipmap Settings**, you can specify the following.

![Mipmap Settings](image)

**Enable**

Enable or disables mipmap generation for the current texture.
<table>
<thead>
<tr>
<th>Description Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maintain Alpha Coverage</strong></td>
</tr>
<tr>
<td>manual adjustment of alpha channel mipmaps is allowed or not.</td>
</tr>
</tbody>
</table>
### Description

**Settings**

**Alpha**

Alpha map's

**Bias**

be multiplied with

a scale based on alpha coverage.

This value will be applied to the final results as an offset.

You can only set this setting if **Maintain Alpha Coverage** is set.

Valid values: 0 to 100
### Mipmap

**Description**

(pixel) is generated when sampling mipmaps.

You can specify the following options:

- **Max**
- **Min**
- **Sum**

**Filter Method**

Specifies the filter method used to process the mipmap.

You can specify the following options:

- **Point**
- **Average**
- **Linear**
- **Bilinear**
- **Gaussian**
- **BlackmanHaris**
- **KaiserSinc**

### Migrate RC.ini Settings to the Texture Settings Editor

The Resource Compiler uses the `rc.ini` file to store your default texture presets. In Lumberyard 1.18, the Texture Settings Editor uses the `ImageBuilderDefaultPresets.settings` instead.

If you want to transfer your settings from the Resource Compiler to the Texture Settings Editor, you must do this manually.
Example

In rc.ini, settings are ordered by preset, which contains the format for all devices. For example, the Albedo preset is stored.

```
[Albedo]
pixelformat=BC1
pixelformat:es3=ETC2
pixelformat:ios=PVRTC4
maxtexturesize:es3=512
rgbweights=ciexyz
powof2=1
colorspace=sRGB,auto
filemasks=_diff*
```

The ImageBuilderDefaultPresets.settings file is in a format that Lumberyard can serialize.

The following example contains the Albedo preset for PC with only the relevant parts that match the settings used in rc.ini. If you modify the PC Albedo preset, you should also make the change to other platforms. This ensures that your changes take effect on all platforms.

```
<ObjectStream version="3">
  ...
  <Class name="AZStd::string" field="value1" value="pc" type="{03AAAB3F-5C47-5A66-9EBC-D5FA4DB353C9}"/>
  ...
  <Class name="PresetSettings" field="value2" version="1" type="{935BCE3F-9E76-494E-9408-47C937D7286B}">
    <Class name="AZ::Uuid" field="UUID" value="{08A95286-ADB2-41E4-96EB-DB48F4726D6A}" type="{E152C105-A133-4D03-BBF8-3D4B2FBA3E2A}"/>
    <Class name="AZStd::string" field="Name" value="Albedo" type="{03AAAB3F-5C47-5A66-9EBC-D5FA4DB353C9}"/>
    <Class name="unsigned int" field="RGB_Weight" value="2" type="{43DA906B-7DEF-4CA8-9790-854106D3F983}"/>
    <Class name="unsigned int" field="SourceColor" value="1" type="{43DA906B-7DEF-4CA8-9790-854106D3F983}"/>
    <Class name="unsigned int" field="DestColor" value="2" type="{43DA906B-7DEF-4CA8-9790-854106D3F983}"/>
    <Class name="AZStd::vector" field="FileMasks" type="{99DAD0BC-740E-5E82-826B-8FC7968CC02C}">
      <Class name="AZStd::string" field="element" value="*_diff*" type="{03AAAB3F-5C47-5A66-9EBC-D5FA4DB353C9}"/>
    </Class>
    <Class name="AZStd::string" field="PixelFormat" value="BC1" type="{03AAAB3F-5C47-5A66-9EBC-D5FA4DB353C9}"/>
    <Class name="bool" field="IsPowerOf2" value="true" type="{A0CA880C-AFE4-43CB-59AC04961127}"/>
    <Class name="AZStd::unique_ptr" field="MipmapSettings" type="{2CB98ED6-5513-BB1-8E9A591BB81}"
      class="MipmapSettings" field="element" version="1" type="{9293961B-23A6-43CB-9887-50528CBFA6FF}">
      <Class name="unsigned int" field="MipGenType" value="5" type="{43DA906B-7DEF-4CA8-9790-854106D3F983}"/>
      <Class name="Color" field="BorderColor" value="0.0000000 0.0000000 0.0000000 1.0000000" type="{7894047A-9050-4F0F-901B-34B1A029417}"/>
      <Class name="unsigned int" field="StreamableMips" value="0" type="{43DA906B-7DEF-4CA8-9790-854106D3F983}"/>
    </Class>
  </Class>
</ObjectStream>
```
Texture Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When creating textures, consider the following best practices and guidelines:

- Use the fewest number of textures that will do the job.
- For road textures, make sure the texture is horizontal.
- Use detail maps to add detail and crispness to lower-resolution textures. Detail maps can be used to add extra grain to wood, extra cracks to a concrete wall, or small scratches to car paint.
- Reuse normal maps and specular maps when possible to save texture memory. Normal maps are twice as expensive memory-wise compared to regular textures. For example, when using several types of floor tiles, brick walls, and concrete walls, create textures so they can use the same normal map and specular map.
- Combine textures for small generic items such as pipes and railings to save on drawcalls. For example, a house can consist of a wall texture, roof texture, and a detail sheet with all windows, frames, and doors. This saves on materials and drawcalls.
- Do not make textures bigger than they appear onscreen. A roof texture on a tall building that neither the player nor the camera can see at close range should be smaller, for example, than a ground texture.
- Use decals to break up and compensate for lack of texture amount. Dirt and stain decals are an easy way to break up tiled textures.
- Use vertex colors to create variety, depth, and color variations. Vertex painting and pre-baked vertex lighting is a relatively cheap way to add depth to objects and make them more interesting.
- Use grayscale textures that can be color-tinted to save on texture memory. Objects that benefit from this technique include cars, fences, barrels, and crates.

Working with Diffuse Maps

When light hits a surface, it splits into two directions: some is reflected immediately off the surface while the rest enters the surface and gets refracted. The refracted light can be absorbed or scattered underneath the surface and exit again at a different angle. This absorbed and refracted light is the diffuse color of an object.

The diffuse color defines how bright a surface is when lit directly by a white light source with an intensity of 100%. Physically speaking, it defines what percentage for each component of the RGB spectrum does not get absorbed when light scatters underneath the surface.

Texture mapping the diffuse color is like applying an image to the surface of the object. For example, if you want a wall object to be made out of brick, you can choose an image file with a photograph of bricks. A diffuse map is always required for objects.

The diffuse map should not contain any lighting, shading or shadowing information, as all this gets added dynamically by Lumberyard. In certain cases, pre-baked ambient occlusion (AO) is required, which is stored in a dedicated AO map in the diffuse channel of the Detail Map. For more information, see Working with Detail Maps (p. 1751).
Diffuse maps can be combined with other texture maps, such as ambient occlusion maps and cavity maps, to create more definition.

**Diffuse Mapping Best Practices**

- Don't use too light or too dark of a texture that will require too much color compensation.
- Metal objects should have a black diffuse color. Rusty metal however needs some diffuse color.
- Paint, or use occlusion mapping, to darken cracks and holes.
- Use crisp colors and contrast to define variations in shapes in order to break up the image.
- Create UV maps so that there is a decent compromise of space utilization and stretching.

**Working with Normal Maps**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The illusion of extra depth and detail to objects is achieved by using normal maps, which are a type of bump map. Bump maps and normal maps both add detail without increasing the number of polygons. As such, they are used to "fake" depth and details such as wrinkles, scratches and beveled edges. Unlike displacement mapping, normal maps affect shading and not the surface itself. The surface remains flat when seen from an angle.

Bump maps store an intensity that represents the relative height (bump) of pixels from the viewpoint of the camera. Traditional normal maps, in addition to storing the height, also store the direction of normals in the RGB values of the texture image. As such, they are more accurate than bump maps.

Lumberyard uses a form of normal mapping, called Tangent Space Normal Mapping, which uses either a height map or is derived from a high-polygon model. In a normal map, a color represents a certain normal vector (surface orientation of a point). For tangent space normal maps the information is relative to the underlying surface.

Tangent space normal maps are independent of the underlying geometry which means the texture can be used on other geometry as well. It will automatically align to the surface regardless of mirroring, rotation, scale or translation. Only the latter two are supported by traditional (object or world) normal maps.

An advantage of tangent space normal maps is that the normals are always pointing outwards, so assuming unit length, the normal z coordinate can be reconstructed from the x and y components. After the coordinate expansion from 0..1 to the -1..1 range, the z component can be computed in the shader with this formula: \( z = \sqrt{1 - x^2 + y^2} \). This makes it possible to use two-channel textures (2 bytes per texel) to store normal maps.

**Topics**

- Normal Mapping Best Practices (p. 1749)
- Using Normals with Gloss Maps (p. 1750)

**Normal Mapping Best Practices**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To get the best results, use the following best practices when creating normal maps.
You can use the following practices in your content creation tool to get the best results from your normal maps:

- Render normal maps using the following swizzle coordinates:
  - +X
  - -Y
  - +Z
- To minimize errors generated by mipmap levels, set the UV padding in your DCC tool to a minimum of 3 pixels.
- When resizing a normal map, use bi-linear interpolation and not bi-cubic.
- When you create mirrored UVs, check the seams and make sure the UVs are facing the same direction on both sides.
- To prevent unexpected lighting issues, make sure the normal map was not altered by the following:
  - Bi-cubic filter
  - Sharpening
  - Alpha blending

Changes to these settings can negatively impact the values in a normal map.

- Preview the normal map in the Material Editor to check for rendering issues.
- For naming conventions:
  - Normal maps with a gloss map in the alpha channel should use the suffix _ddna. For example, road.tif would be road_ddna.tif.
  - Normal maps without a gloss map in the alpha channel should use the suffix _ddn. For example, road.tif would be road_ddn.tif.

Using Normals with Gloss Maps

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Most materials should have a gloss map as well as a normal map as this can impart a lot of variation to the shading. Gloss maps are closely related to normal maps, as high frequency details in a normal map can create some roughness as well. However, gloss is more the microscale roughness of the material while normal represents macro-scale bumpiness. Gloss maps are treated like diffuse maps.

The Gloss map always goes into the Alpha channel of the Normal map, even if you’re using a specular map for metals and metal-embedded surfaces.

If the preset NormalMapWithGlossInAlpha_highQ is selected, the Resource Compiler will automatically adjust the gloss map stored in the alpha channel based on the normal variance and lower the gloss where normals are very bumpy. This can greatly help to reduce shimmering and sparkling highlights artifacts.

Lumberyard uses DDNA textures, which is a standard DDN texture with the addition of a Gloss map in the alpha channel of the normal map. DDNA texture map must use the _ddna.dds filename suffix (instead of _ddn.dds) for the Resource Compiler to recognize the texture correctly.

Working with Gloss Maps

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Gloss defines the roughness of a surface. A low gloss value means that the surface is rough while a high value means the surface is smooth and shiny. The roughness influences the size and the intensity of specular highlights. The smoother and glossier a surface is, the smaller the specular highlight will be. A smaller highlight will at the same time be brighter in order to obey to the rules of energy conservation.

For physically-based shaders, the gloss map is highly important. Most materials should have a gloss map as well as a normal map as this can impart a lot of variation to the shading. Gloss maps are closely related to normal maps, as high frequency details in a normal map can create some roughness as well. However, gloss is more the microscale roughness of the material while normal represents macro-scale bumpiness. Gloss maps are treated like diffuse maps.

The Gloss map always goes into the Alpha channel of the Normal map, even if you're using a specular map for metals and metal-embedded surfaces.

Gloss mapping is more powerful than the traditional specular mask, as gloss influences not only the brightness of a highlight but also its size and the sharpness of reflections.

When working with textures, gloss maps and normal maps are created first, then diffuse maps. Diffuse maps should contain no lighting information.

The gloss map is always stored in the alpha channel of the normal map. If the preset NormalMapWithGlossInAlpha_highQ is selected, the Resource Compiler will automatically adjust the gloss map stored in the alpha channel based on the normal variance and lower the gloss where normals are very bumpy. This can greatly help to reduce shimmering and sparkling highlights artifacts.

**Gloss Map Best Practices**

- Put variation into the gloss map. Not just random noise but really where the object would be less or more rough.
- If an object has the correct physical specular color but does not show specular highlights on top of the diffuse, the gloss is likely set too low. Increase the brightness of the gloss map.
- The Glossiness value must be set to 255, otherwise the gloss map will not work.
- Non-metals should have a specular color value between 53 and 61, based on what looks the best.
- For metals (and for metal parts embedded in non-metals), a dedicated specular texture map is used, with the gloss map going into the alpha channel of the normal map. The gloss map defines the smoothness, reflectivity and tightness of specular highlights. For metals, the shader doesn't control specular color – the specular texture map does. Specular color is physically based. Because of this, set the Specular color value to 255.

**Working with Detail Maps**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Detail mapping is a simple technique to add macro surface detail at relatively low cost, memory and performance wise. The following best practices should be taken into consideration:

- Use as low a resolution as possible for best performance (512x512 or lower).
- Prevent artifacts by using a higher tiling scale.
- Decrease contrast for the detail diffuse and gloss.

Unified detail mapping (UDM) is basically a reversed detail map. Usually the detail map is used for finer details as you get closer. UDM is the opposite. It helps to define big shapes viewed from the distance.
Since close-up detail is provided in tiled textures, larger details are needed to define shapes better when viewed from a distance.

**Setting Up Detail Map Textures**

Detail map parameters are setup in the Material Editor.

**To set Detail Map parameters**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the left tree, select an applicable texture.
3. In the right pane, under **Shader Generation Params**, click the **Detail Mapping** check box.
4. Under **Shader Params**, set values for the following parameters.
   - **Detail bump scale**: Defines how much the normal map is visible. The higher the value, the more the normal map will show through.
   - **Detail diffuse scale**: Defines how much the diffuse map (or AO map) visible. The higher the value, the more the normal map will show through.
   - **Detail gloss scale**: Defines how much the gloss map is visible. The higher the value, the more the gloss map will show through.

**Working with Decals**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Decals are non-repeating images or textures that are applied to the surface of an object or terrain with a specified projection. Common examples of decals are product labels and logos, artwork for walls, signs, and surface cracks.

Decals can break up uninteresting textures and bring together such level elements as brushes and terrain. Good decal placement can also create seamless transitions between many different objects. Decals only work with the **Illum Shader** (p. 1697).

**Note**

If you apply decals to an object that can be moved by a player, the decal will not move with the object.

**Topics**

- Decal Projection Types (p. 1752)
- Placing a Decal (p. 1753)
- Setting Decal Parameters (p. 1754)
- Debugging Decal Mapping Issues (p. 1757)

**Decal Projection Types**

Decals have several different projection types. To change projection type, select the decal and change the **ProjectionType** value.
Planar Projection

Planar projection is the cheapest performance-wise. The decal is displayed in the same location as the center of the object. Use planer projection only on flat surfaces, otherwise the decal may appear to be floating.

Deferred Projection

Deferred projection is a simple method to get decals to follow the contours of objects and is similar to Planar projection, but slower. As such, use Planar projection wherever possible.

Deferred projection is enabled by selecting the Decal Params, Deferred check box.

ProjectOnTerrain Projection

The decal is projected directly onto the terrain, ignoring any objects that might otherwise receive the projection.

ProjectOnStaticObjects Projection

The decal is projected onto the geometry of an object along the opposite direction of the blue z-axis. This method is automatically done as a deferred pass.

ProjectOnTerrainAndStaticObjects Projection

A combination of ProjectOnStaticObjects and ProjectOnTerrain, the decal is displayed on both the terrain and on objects. This method is automatically performed as a deferred pass.

Placing a Decal

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Do the following to place a decal in your level.

Note
If you apply decals to an object that can be moved by a player, the decal will not move with the object.

To place a decal

1. In Lumberyard Editor, click the top Follow Terrain button.
2. In Rollup Bar, click Objects, Misc, Decal.
3. Drag the detail into the level and then click to place it.
4. Using the Edit menu, move, rotate, or scale the decal as needed.
5. To place a decal manually, select the Reorientate check box, and use mouse shortcuts to place the decal as follows. This can speed up placement enormously.
a. Ctrl+Click: Move the decals to the desired position
b. Alt+Click: Scales the decal along the X, Y axes
c. Ctrl+Alt+Click: Rotates the decal around the z-axis

To place a decal on vegetation

1. Enable deferred projection so the decal follows the contours of the vegetation:
   a. In Lumberyard Editor, in the Rollup Bar, under Objects, click Misc, Decal.
   b. Under Decal Params, select the Deferred check box. For information about projection types, see Decal Projection Types (p. 1752).

2. Enable the r_deferredDecalsOnDynamicObjects console variable so the decal appears on the vegetation:
   a. In Lumberyard Editor, click the X icon in the Console section.

   ![Console](image)

   b. In the Console Variables window, search for r_deferredDecalsOnDynamicObjects
   c. Set the value to any positive number, for example 1.
   d. Close the Console Variables window to save the new value.

3. Follow the instructions above for placing a decal.

Setting Decal Parameters

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Complete the following procedures for setting decal mapping parameters.

To set decal parameters in the Rollup Bar

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Most decal parameters are in the Rollup Bar on the Objects tab, under Misc, Decal.
To set decal parameters in the Rollup Bar

1. In the Rollup Bar, under Objects, click Misc, Decal.
2. Under Decal Params, adjust the following parameters:
- **ProjectionType** – Choose the projection type from the drop-down list: Planar, ProjectOnStaticObjects, ProjectOnTerrain, and ProjectOnTerrainAndStaticObjects.
- **Deferred** – Select to enable deferred decal projection.
- **View Distance Multiplier** – Set the distance at which the decal is visible. The default value is 1. A higher number indicates a longer visibility distance.
- **SortPriority** – Specify if the decal will appear on top of another decal.
- **Projection Depth** – Set the projection depth (distance) of the decal from the object. This can also affect blending of decals.

**To set shader decal parameters**

Some decal parameters are set in the Material Editor under **Shader Params**.

1. In Lumberyard Editor, click **Tools**, **Material Editor**.
2. Click the **Add New Item** button.
3. Select a decals folder, select a subfolder, and then click **Save**. The new material will be selected automatically with the default settings.
4. Under **Shader Generation Params**, select **Decal**.
5. Right-click the decal you created and click **Assign to Selected Objects**.
6. Under **Shader Params**, adjust the values of the following parameters:
   a. **Decal Alpha Falloff** – Set the power applied to the decal alpha.
   b. **Decal Alpha Multiplier** – Set the multiplier applied to the decal alpha.
   c. **Decal Diffuse Opacity** – Set the opacity multiplier for the fading out decal diffuse color.
Debugging Decal Mapping Issues

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

See the following ways you can debug decals.

Debugging Deferred Decals

The cost of a deferred decal depends on how many objects the decal projects, how expensive the geometry is, and how many overdraws the decal creates.

You can use the console variable (CVAR) `r_deferredDecalsDebug` to show how expensive it is to render a deferred decal. Specify a value of 1. For more information, see Using the Console Window (p. 210).

Deferred decals in the Lumberyard Editor viewport render in red, green and blue. The colors show you the expense for rendering a deferred decal.

- Red = expensive
- Green = medium
- Blue = cheap

We recommend that you place deferred decals so that they display mostly in blue.

Debugging Decal Flicker

If a placed decal is flickering, follow these guidelines to ensure that the decal is correctly set up.

- In the Material Editor, for the Shader Generation Params pane, verify that the Decal parameter is selected for all submaterials.
- Check for overlapping layers that have the Decal parameter selected. Use the Sort Priority parameter to specify which decal appears on top of the other. Decals with higher Sort Priority values sort on top of decals with lower values. For more information, see the Decal Component Properties (p. 585).
- Other than for decals, the mesh shouldn't have overlapping triangles. Do not offset along the surface normal; this can still break in some situations and introduce floating parallax effects.

Displacement Maps and Tessellation

Displacement mapping allows you to displace the actual surface geometry of an object to give you extra depth and detail than is available using bump mapping, offset bump mapping or parallax occlusion mapping (POM) techniques, which all "fake" surface detail. Displacement mapping results are dependent on how far the camera is from the object.

Displacement mapping uses a texture map, called the height map, which is used to define the value of vertex height displacement. Specifically, this is a scalar displacement that is stored in the alpha channel of a *.displ* texture file.
In order for displacement mapping to work correctly, you need to also apply tessellation to your object, otherwise there wouldn't be enough geometry to displace. Tessellation increases the geometry count by subdividing polygons into smaller polygons before it gets displaced.

**Topics**
- Displacement Mapping Best Practices (p. 1758)
- Setting Displacement Mapping Parameters (p. 1758)
- Tessellation (p. 1759)

**Displacement Mapping Best Practices**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For best results, use the following best practices when creating displacement maps.

**Content Creation Practices**
- Place the displacement map in the alpha channel of your texture. The RGB values are not used for this map type and can be left empty.
- Use the suffix _displ. For example, road.tif would be road_displ.tif.

**Game Implementation Practices in Lumberyard**
- In Lumberyard Editor, set the performance to High or Very High for your target platform.
  
  **To set the graphics performance for your target platform**
  
  1. In Lumberyard Editor, choose Edit, Editor Settings, Graphics Performance.
  2. Select your platform and then select the performance setting.
- The console variable e_ShadowsTessellateCascades=1 enables shadows for tessellated geometry. Use this console variable sparingly, as this feature increases performance costs.
- Use the Displacement Map preset to store _displ textures. Height maps are converted to A8 textures. If you don't see any displacement, open the texture file in the Material Editor and verify the rendering in preview. If the file isn't in A8 format, fix the preset, save, and reload.
- In the Material Editor, under Shader Generation Params, select the Displacement mapping parameter. Under Texture Maps, for the Height Map parameter, assign the displacement map.

**Setting Displacement Mapping Parameters**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**To apply displacement mapping to an object**

1. In Lumberyard Editor, click Tools, Material Editor.
2. In the left tree, select the desired asset.
3. In the right pane, under Shader Generation Params, select Displacement mapping.
4. Under **Shader Params**, adjust the values of the following parameters for the desired effect.
   
a. **Displacement bias** – Moves the plane where the displacement is applied. This reduces gaps in meshes, and prevents objects from displacing other objects that are placed above them.

b. **Displacement height scale** – Changes the overall height of the displacement.

**Tessellation**

In order for displacement mapping to work correctly, tessellation is also required, otherwise there wouldn't be enough geometry to displace. Tessellation increases the geometry count by subdividing polygons into smaller polygons before it gets displaced. Phong and PN triangles are the two available tessellation methods.

**Phong tessellation** approximates smoothing based on surface normals. Surfaces with Phong tessellation applied are not perfectly smooth across patch boundaries, causing the object to look inflated.

**PN triangle tessellation** is similar to Phong tessellation and is slower, but with better approximation.

Tessellation is only supported for the Illum Shader (p. 1697) and HumanSkin Shader (p. 1696).

**Topics**

- Setting Tessellation Parameters (p. 1759)
- Fixing Tessellation Seams (p. 1760)

**Setting Tessellation Parameters**

To apply tessellation to an object and set parameter values, complete this procedure.

**To apply tessellation to an object**

1. In Material Editor, click **Tools, Material Editor**.
2. In the left tree, click to select the desired asset.
3. In the right pane, under **Shader Generation Params**, select either **Phong tessellation** or **PN triangles tessellation**.
4. Under **Shader Params**, adjust the values of the following parameters.

**Tessellation Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tessellation face cull</td>
<td>Specifies the extent to which vertices are culled. Because tessellation uses its own face culling, it takes the original (non-tessellated) triangle and checks if it's facing the camera; if not it discards it.</td>
</tr>
</tbody>
</table>
Parameter | Description
--- | ---
| This can also be used for 2-sided sorting of polygons. In this case, the 2 Sided check box must also be selected under Advanced in the Material Editor. An issue may arise when there is displacement that is visible from the camera. For example, a bump on a cube that is rotating is still visible for a while, even though the cube face is no longer facing the camera. Setting this parameter to 0 means no face culling at all, while setting it to 1 will cull anything not facing the camera. |

Tessellation factor | Specifies the density of the mesh triangles
Tessellation factor max | Used for objects that are at a fixed distance or range from the camera to get rid of geometry “popping” artifacts. This is useful for cutscenes.
Tessellation factor min | Setting this value to 1 means that it will be always tessellated at level 1, even if the object is far away from camera.

Fixing Tessellation Seams

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

There are two types of seams or cracks that can become noticeable when using tessellation.

**Border Seams**

Border seams occur when different meshes are placed close to each other, or when a mesh consisting of sub-meshes causes unpleasant cracks because of using different materials with different displacement (or even same displacement maps with slightly different UV mapping).

The solution involves carefully placing meshes or fade-out displacement by modifying the displacement map as needed.

**UV Seams**

UV seams occur when two adjacent triangles share an edge but use separate vertices with different UVs. This shared edge will have a different displacement on each side due to sampling different places in the displacement map. Even tiny differences in UV can cause visible seams. This is automatically fixed by Lumberyard if there is no tiling. Otherwise you must change the UV mapping to hide such artifacts where possible.

Phong tessellation and PN Triangle tessellation do not suffer from UV seams as they do not use UV mapping.

**Working with Substances**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Substances are procedural materials created using Allegorithmic's Substance Designer. Lumberyard has the ability to import Substance .sbsar files using the Substance Editor.

**Creating Substances for Lumberyard**

When creating Substances for Lumberyard using Allegorithmic's Substance Designer, it is recommended to use the PBR Specular/Glossiness substance as the base. This will involve less adjustments to your default outputs for substances. However, you will need to delete the Glossiness output and save the Gloss map into the alpha channel for the Normal map output in Substance Designer.

If you want to use a PBR Metallic/Roughness Substance and convert it for use in Lumberyard, follow these steps:

- Change the BaseColor output node to Diffuse.
- Create a Specular output node in the Substance Graph.
- Create a RGB-A Merge node in the Substance Graph.
  - Connect the node that was originally going into the Normal map into the RGB input.
  - Note that the A (Alpha) input for this node will be connected later on.
  - Connect the output of this merge node into the input for the Normal output node.
- Create a BaseColor/Metallic/Roughness converter node in the Substance Graph.
  - Connect the node that was originally going into the BaseColor/Diffuse map into the BaseColor input for this converter node.
  - Connect the node that was originally going into the Metallic map into the Metallic input for this converter node.
    - Connect the Diffuse output of this converter node into the input for the Diffuse output node.
    - Connect the Specular output of this converter node into the input for the Specular output node.
    - Connect the Glossiness output of this converter node into the A (Alpha) input for the RGB-A Merge node.
- Delete the Roughness output node.
- Delete the Metallic output node.
- Save the changes to your Substance and then publish the .sbs as a .sbsar to be imported into Lumberyard.

**Exporting Substances for Lumberyard**

When using Allegorithmic's Substance Designer to export textures to Lumberyard, take extra care to select the correct file type. The following file types export correctly: .tif, .tga, .bmp. Avoid exporting with the following common file types: .png, .jpg, .psd.

**To export textures in Substance Designer for Lumberyard**

1. In the File menu, click Export Textures.
2. Click the Configuration tab, and then under Presets, select Amazon Lumberyard.
3. Click the Export tab. In the file type drop-down menu (upper right corner), select one of the following file types:
   - .tif
   - .tga
   - .bmp
4. Click **Export**.

**Working with Substance in Lumberyard**

Using Substance Editor, you can edit Substance material properties and visualize substances on objects in real-time. Substance Editor also has the ability to generate and export static textures from Substances.

Here are some things to keep in mind when working with Substances in Substance Editor:

- The Substance Gem needs to be enabled first for the project using *Creating Lumberyard projects* (p. 43). For more information on Gems, see *Add modular features and assets with Gems* (p. 1064).
- When importing substance files, you must restart Lumberyard Editor before substance textures are rendered correctly.
- A `.smtl` (substance material) file and a `.sub` (substance texture) file are generated in the same directory location as the imported `.sbsar` for applying the substance material or substance textures to objects.
- By default, an `.smtl` file will inherit the `.sub` files in the appropriately matching channels based on the outputs in the published `.sbsar` from Substance Designer. For example, a diffuse output texture will map into the diffuse channel for the `.smtl` file.

**To use Substance Editor**

1. Open Lumberyard Editor and select **Tools, Plug-Ins, Substance Editor**. You can also click the Substance icon in the main toolbar of Lumberyard Editor.
2. To update imported `.sbsar` files, click **Edit, Reimport Substance**. Current changes will not be overwritten.
3. To remove a substance, click **File, Delete Substance**.

  **Note**
  This permanently removes the substance and all associated assets from the `.sbsar` project, which cannot be recovered using the Windows Recycle Bin.
Console Variables for Substances

To control how Substances are handled by CPU and memory, you can set the following CVars in your `system_windows_pc.cfg` file or `editor.cfg` file.

**substance_engineLibrary**

Enables substances to run on the GPU instead of the CPU, permitting the output of higher resolution textures. Specifies the engine to load for substance plugin. Possible values are PC: `sse2|d3d10|d3d11`. The default is `sse2`, which specifies CPU.

**substance_commitRenderOptions**

Applies CPU and memory changes immediately, rather than waiting for the next render call.

**substance_coreCount**

Sets how many CPU cores are used for Substance rendering. A value of 32 specifies all cores. This setting is relevant only when using CPU based engines.

**substance_memoryBudget**

Sets, in megabytes, the amount of memory used for Substance rendering. The default is 512.

Working with Blend Layers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Blend layers can be thought of as a layered material. You can create one material with a second set of per-pixel functions, including texture maps, UV tiling and offset, rotation and oscillation animation, second UV set, diffuse and specular color tinting, smoothness, and blend masking.

Integrate blend layers in the Illum and Vegetation shaders and activate them by setting the blend layer shader generation parameters.
Working with Blend Layers

Topics
- Shader Generation Parameters (p. 1764)
- Blend Layer Textures (p. 1764)
- Blend Layer Parameters (p. 1765)

Shader Generation Parameters

You can set the following parameters for generating shaders.

**Blend layer**
- Activate the texture inputs and parameters for the blend layer.

**Use UV set 2 for blend layers maps**
- Use a second UV channel, if available, for the second layer.

Blend Layer Textures

You can set the following parameters for your blend layer textures.

**Second Height Map**
- Displacement map for the blend layer (grayscale)

**Second Diffuse Map**
- Diffuse map for the blend layer (RGB)

**Second Normal Map**
- Normal map for the blend layer; the second layer gloss map is contained in the alpha (RGBA)
Second Specular Map
Specular map for the blend layer (RGB)

Blend Map
Blending map to blend between the first and second layers (grayscale)

Blend Layer Parameters
You can set the following parameters for your blend layer.

Blend Factor
Control visibility of the blend layer, with a bias toward black or white
Valid values: 0 – 16

Blend Falloff
Control the fall off of the blend range
Valid values: 0.1 – 128

Blend Layer 2 Tiling
Control tiling of the second blend layer
Valid values: 0 – 20
Default: 1

Blend Layer 2 Diffuse (Tint)
Base color tint for the second layer (RGB)
Valid values: 0 – 255

Blend Layer 2 Smoothness
Sharpness of specular reflection for the second layer
Valid values: 0 – 255
Default: 10

Blend Layer 2 Specular
Reflective brightness and color for the second layer (RGB)
Valid values: 0 – 255

Blend Mask Tiling
Control tiling of the blend map
Valid values: 0.05 – 20
Default: 1

Parallax Mapping

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Parallax occlusion mapping (POM) is an enhancement of the traditional parallax mapping technique that creates detail in a texture to add the illusion of depth. This depth perception changes based on perspective.

Parallax occlusion mapping (POM) and offset bump mapping (OBM) are similar to displacement mapping and tessellation, but their computational requirements are not as demanding. However, POM is not suitable in every situation. Use POM only for high-performance computers and OBM for devices such as consoles. When you use POM, you must enable both shader generation parameters. Lumberyard automatically defaults to OBM for setups that cannot run POM.

Topics
- Parallax Mapping Best Practices (p. 1766)
- Applying Parallax Occlusion Mapping (POM) (p. 1767)
- Applying Silhouette Parallax Occlusion Mapping (SPOM) (p. 1767)
- Using Blend Layers for Parallax Mapping (p. 1768)

Parallax Mapping Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For best results, use the following best practices when creating POM or SPOM parallax mapping.

Content Creation Practices
- Place the displacement map in the alpha channel of your texture. The RGB values are not used for this map type and can be left empty.
- Use the suffix _displ. For example, road.tif would be road_displ.tif.
- If your project requires file names other than those listed here, use the Resource Compiler (RC) Open Image tool to specify other conventions for Asset Processor to recognize. For textures not using the _displ suffix, choose the DisplacementMap preset to generate the correct .dds file. Height maps are converted to A8 textures. If you do not see any displacement, preview the file in the Material Editor. If the image isn't in A8 format, you can fix the preset, save, and reload. For textures using non-
standard naming, the tool generates an .exportsetting file that Asset Processor uses to generate the displacement map.

- By default, Asset Processor recognizes the _displ suffix and writes the correct metadata to a .dds file.

**Game Implementation Practices in Lumberyard**

- In Lumberyard Editor, set the performance to **High** or **Very High** for your target platform.

  **To set the graphics performance for your target platform**

  1. In Lumberyard Editor, choose **Edit, Editor Settings, Graphics Performance**.
  2. Select your platform and then select the performance setting.

- In the **Material Editor**, you must first assign textures for both the **Normal Map** and the **Height Map** parameters. Then, under **Shader Generation Params**, select the **Parallax occlusion mapping** parameter.

- You can assign POM maps to the **Height Map** parameter located under **Texture Maps**.

**Applying Parallax Occlusion Mapping (POM)**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To apply POM, complete the following procedure.

**To apply Parallax Occlusion Mapping**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the left tree, select the desired asset.
3. In the right pane, under **Shader Generation Params**, select **Offset bump mapping** and **Parallax occlusion mapping**.
4. Under **Shader Params**, adjust the values of the following parameters.
   a. **Height bias** – Moves the plane where the displacement is applied. This reduces gaps in meshes, and prevents objects from displacing other objects that are placed above them.
   b. **POM Displacement** – Sets the POM depth. A larger value adds more depth.
   c. **Self shadow strength** – Changes the strength of self-shadowing. A larger value imparts more shadowing
5. Under **Texture Maps**, enter the paths to the various textures.

**Applying Silhouette Parallax Occlusion Mapping (SPOM)**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To apply SPOM, complete the following procedure.

**To apply Silhouette Parallax Occlusion Mapping**

1. In Lumberyard Editor, click **Tools, Material Editor**.
2. In the left tree, select the desired asset.
3. In the right pane, under Shader Generation Params, select Parallax occlusion mapping with silhouette.
4. Under Shader Params, adjust the values of the following parameters.
   a. **Height bias** – Moves the plane where the displacement is applied. This reduces gaps in meshes, and prevents objects from displacing other objects that are placed above them.
   b. **Self shadow strength** – Changes the strength of self-shadowing. A larger value imparts more shadowing.
   c. **Silhouette POM Displacement** – Sets the SPOM depth. A larger value adds more depth.
5. Under Texture Maps, enter the paths to the various textures.

### Using Blend Layers for Parallax Mapping

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use blend layers for parallax mapping. For both POM and OBM, set the diffuse and normal map as usual. The \_disp texture will be loaded automatically as long as the Applying Parallax Occlusion Mapping (POM) (p. 1767) procedure is first completed.

When using a second blend layer, the diffuse map is placed in the Custom texture map slot, the normal map is placed in the [1] Custom slot, and the height map is placed in the SubSurface slot.

**To use a blend layer for parallax mapping**

1. Complete the Applying Parallax Occlusion Mapping (POM) (p. 1767) procedure.
2. Under Shader Generation Params, select Parallax occlusion mapping and Blendlayer.
3. Under Texture Maps, place maps as follows:
   a. Place the height map in Second Height Map.
   b. Place the height map in Second Diffuse Map.
   c. Place the height map in Second Bump Map.
4. Under Shader Params, adjust the values of the parameters as needed.

### Using Vertex Colors

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Vertex color, or vcolor, is just a color with RGB and alpha channel values stored for each vertex of a mesh. Vertex color and alpha can be used for multi-texturing, transparency, or fake ambient occlusion.

Vertex color is typically multiplied against the Diffuse color, colorizing or darkening the color map.

When used for non-color effects, typically each color channel is treated as a separate monochrome set of values, so for example vertex color can control three different per-vertex effects.

Vertex Colors is a Shader Generation parameter that can be enabled using the Material Editor, which is part of Lumberyard Editor.
Customizing Post-Processing Effects

For a good application of vertex colors, see Defining Vegetation Vertex Colors with a DCC Tool (p. 1307).

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes post-processing effects that can help improve your game's graphics, lighting, and transitions between effects such as color correction, bloom, and depth of field.

Use XML files with Script Canvas editor or Lua scripts to customize effects by setting their parameters, create prioritized groups of effect parameter sand enable or disable them.

You can also use effect groups to specify the following:

- Blend curves to smoothly transition between effects
- Stay enabled until explicitly disabled
- Make effect strength based on distance from the camera

**Note**

Creating a new effect requires modifying Lumberyard, while creating a new effect group does not.

**Topics**

- Post Effect Group XML Files (p. 1769)
- Enabling and Disabling Effect Groups (p. 1770)
- Specifying a Blend Curve for Smooth Effect Transitions (p. 1771)
- Setting Effect Strength Based on Camera Distance (p. 1772)
- Using the Screen Fader Effect (p. 1773)

**Post Effect Group XML Files**

When you open Lumberyard Editor, the effect group located at `lumberyard_version\dev\Engine\Libs\PostEffectGroups\Default.xml` automatically loads. The `Default.xml` file includes all available effects and the default values for each parameter. You can modify the default values and copy and paste sections of the `Default.xml` file into custom effect groups.

**Example**

```xml
<PostEffectGroup priority="1" hold="1">
    <Effect name="Global">
        <Param name="User_Brightness" floatValue="0.5"/>
    </Effect>
</PostEffectGroup>
```
Priority

Non-negative integer used to set priorities. Larger priorities override smaller priorities. If multiple effect groups that are enabled have the same priority value, the effect group that was enabled later has the higher priority.

Default value: 0 (for Time of Day and Script Canvas nodes that set effects)

Valid values: 0 - 999

Hold

Indicates if the effect should stay enabled until explicitly disabled.

Default value: 0

Valid values: 0 = effect is disabled after blending is complete | 1 = effect remains enabled until explicitly disabled

If you create a custom effect groups, create a directory named \PostEffectGroups under /Engine/Libs. You can then load the post effect group XML files from any valid path location.

Enabling and Disabling Effect Groups

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can enable and disable effect groups with the Script Canvas editor or Lua scripting.

To enable or disable effect groups with Script Canvas

1. In Lumberyard Editor, open your level.
2. In the menu bar, choose Tools, Script Canvas.
3. In the Script Canvas editor, choose File, New Script.
4. Right-click the canvas, search and then select the following nodes:
   - On Graph Start
   - Enable Effect Group (p. 2622) or Disable Effect Group (p. 2610)
5. For your script, do the following:
   a. Select the Out pin for On Graph Start and drag to connect it to the In pin for the Enable/Disable Effect Group node.
   b. For the Group Name parameter, enter the file path for the group (for example, Libs \PostEffectGroups\MyEffectGroup.xml).
6. Save the script and attach it to an entity with the **Script Canvas (p. 818)** component.
7. Close the **Script Canvas** editor.

**Note**
You can manually enable or disable an effect group in Lumberyard Editor by running the Lua functions in the **Console** window. To indicate a Lua command, prepend each command with the `#` character.

### Specifying a Blend Curve for Smooth Effect Transitions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can use **BlendIn** and **BlendOut** tags to specify a blend curve that enables smooth transitions between effects.

An example XML file with added **BlendIn** and **BlendOut** tags:

```xml
<PostEffectGroup priority="1" hold="1">
  <Effect name="SunShafts">
    <Param name="RaysAmount" floatValue="0.2"/>
  </Effect>
  <BlendIn curve="smooth">
    <Key time="0" value="0"/>
    <Key time="0.5" value="1"/>
  </BlendIn>
  <BlendOut curve="smooth">
    <Key time="0" value="1"/>
    <Key time="0.5" value="0"/>
  </BlendOut>
</PostEffectGroup>
```

**Priority**

Indicates how much the effects should override the lower priority values.

**Hold**

Determines when the **BlendIn** and **BlendOut** curves play and whether the effect group is enabled or disabled.
Valid values:

0 = Plays the BlendOut curve immediately after the BlendIn curve finishes playing; when the BlendOut curve plays, the effect group is disabled

1 = Plays the BlendIn curve; when the BlendIn curve plays, the effect group fully overrides lower priority values until the effect group is explicitly disabled

Curve

Available curve types are smooth, linear, and step. If a curve attribute value is not specified, the curve type defaults to smooth. You can include as many keyframes in a curve as desired.

Default curve value: smooth

Valid key time values: smooth, linear, step

Key time

Valid values: 0 – 1 (seconds)

Setting Effect Strength Based on Camera Distance

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the fadeDistance attribute to set the effect strength based on the distance from the camera.

Example opening XML tag using the fadeDistance attribute:

```xml
<PostEffectGroup priority="1" fadeDistance="20">
```

fadeDistance – Indicates how the effects are actualized based on the distance of the camera from the entity.

- When the camera is at the position of the entity, the effects are fully overridden.
- When the camera is less than fade distance from the entity, the effects are blended.
- When the camera is at least fade distance from the entity, the effects are set to the lower priority values.

Script Canvas Editor

You need to use a script to apply this effect group at the specified position for each frame. For more information, see the Apply Effect Group At Position (p. 2606) node.

Example

The following script uses the Get World Translation to send the entity's position to the Apply Effect Group At Position node; this node then applies the effect group to that specified position.
Lua
To enable an effect group using Lua, set the position at which to apply the effect by using the following function:

```lua
local pos = TransformBus.Event.GetWorldTranslation(EntityId)
PostEffects.ApplyEffectGroupAtPosition("example.xml", pos)
```

**Note**
This function must be called once per frame while the effect group is enabled. If this function is called multiple times in a single frame, the effect strength increases each time, as if each call applies the effect from a different entity.

Using the Screen Fader Effect
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the screen fader effect to control all screen fade properties, including duration of the fade effect and the color or texture to use for the fade.

You can stack multiple screen fader effects; multiple screen faders are rendered concurrently and in order of post effect group priority. Screen fader effects do not blend the parameters between layers.

The following example demonstrates how to use the screen fader effect in an `.xml` file:

```xml
<PostEffectGroup priority="2" hold="1">
  <Effect name="ScreenFader">
    <Param name="Enable" floatValue="1.0"/>
    <Param name="FadeInTime" floatValue="2.5"/>
    <Param name="FadeOutTime" floatValue="1.0"/>
    <Param name="ScreenCoordinates" vec4Value="0.0,0.0,1.0,1.0"/>
    <Param name="FadeColor" vec4Value="0.2,0.7,0.7,0.5"/>
    <Param name="TextureName" stringValue="textures/StyleTown/_dev_Blue_Light.tif"/>
  </Effect>
</PostEffectGroup>
```

**Parameters**
You can set the following parameters:

**Enable**
Determines whether or not the ScreenFader is active for the post-processing effect group.
Valid values: 0 = Disable | 1 = Enable

**FadeInTime**

Time, in seconds, for the screen fader to fade in, once enabled.

**FadeOutTime**

Time, in seconds, for the screen fader to fade out, once disabled.

**ScreenCoordinates**

Determines the rectangle where the screen fader is rendered. Specify the coordinates in the format (Left, Top, Right, Bottom). For example, a fullscreen quad is specified as (0.0, 0.0, 1.0, 1.0) and a quad that fills half of the screen is specified as (0.0, 0.0, 0.5, 1.0).

Valid values: 0.0 – 1.0

**FadeColor**

Sets the quad color by multiplying the color by the specified texture. If no texture is specified, the quad will be a solid color.

**TextureName**

Path of the texture to use for the screen fader.

---

**Toon Shading (Experimental Feature)**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Toon shading is a feature with which you can simulate cartoon effects for your game. Instead of using a shade gradient, toon shading uses less shading color to make 3D graphics appear flat. You can use toon shading to create a comic book or cartoon style for your game. Lumberyard's toon shading feature projects the scene luminance into a lookup table, which controls the smoothness of shading on the surface to achieve the intended look.

As a best practice for toon shading, use clear textures with fewer patterns when designing game art. Also, create models with fewer features at a lower level of detail (LOD). Otherwise, your models can have darker meshes in the distance due to their feature lines. For example, if you have a high density of polygons for vegetation, the vegetation can appear too dark or have too many black lines.

To enable toon shading, use the console to specify the console variable (CVAR):

r_AlphaBlendLayerCount.

Valid values: 0 – 1

For more information, see Configuring Console Variables (p. 212).

The following example level does not have toon shading enabled (r_AlphaBlendLayerCount=0).
The following example level has toon shading enabled (r_AlphaBlendLayerCount=1).

See additional examples of toon shading enabled:
Order-Independent Transparency

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Order-independent transparency (OIT) corrects the display of transparent objects that are drawn out of order.

OIT is useful when creating the following:

- Concave geometry – When you create concave geometry, such as a glass, wine glass, or glass sculpture, some triangles may cover the same pixels and are drawn on top of each other. OIT solves the out-of-order issues that appear from certain angles.
- Intersecting geometry – When you create intersecting geometry, such as hair planes, some triangles may intersect in separate draws. OIT properly orders the triangles for each pixel.
- Transparent objects inside transparent objects – This includes liquids inside of glasses, holograms, or x-ray style effects.
To further demonstrate, see the following images.

| OIT is not needed in the following example. Drawing object 1, then object 2, and finally object 3 works. | OIT is needed in the following example. Drawing object 1 and then object 2 will not work because certain pixels from object 1 must be in front of pixels from object 2 (and vice versa). | OIT is needed in the following example. Object 2 is long and its center point is in front of object 1, but it must draw behind object 1. | OIT is needed in the following example. Certain parts of object 1 draw behind other parts of the object, and you cannot control which draws first. |

Configuring Order-Independent Transparency for Lumberyard

You can configure OIT with the console or create a level configuration file with this setting.

Specify the value for `r_AlphaBlendLayerCount` to the number of transparency layers that OIT can solve. A value of 1 is sufficient to solve most out-of-order transparency issues.

Valid values: 0 – 4

0 = Disables OIT

- Specify the value for the `r_AlphaBlendLayerCount` console variable
- Specify the value for the `level.cfg` file in the `lumberyard_version\dev\project_name\Levels\level_name` directory.

For more information, see Configuring Console Variables (p. 212).
OIT requires the following:

- Hardware requirements: DirectX 12_1 feature level compatible graphics card (NVIDIA Maxwell & Pascal, 4th generation Intel core processors).
- Software requirements: DirectX 11.3 and 12 runtime on Windows 10 compiled with Windows 10 SDK.

To set the Windows 10 SDK for Lumberyard

1. With a text editor, open the `user_settings.options` file in the `lumberyard_version\dev\_WAF_` directory.
2. Under the `[Windows Options]` section, set `win_vs2017_winkit` to your version of the Windows 10 SDK (for example, `10.1.17134.12`).
3. Build your game project. For more information, see Building Lumberyard projects (p. 61).

Adding lighting and shadows

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard uses physically-based lighting and shading models to implement global illumination and lighting.

For information about the Light entity and the Environment Probe entity used in environment lighting, see Light Entities.

For information about using the Time of Day Editor to simulate the changing lighting effects caused by the sun moving across the sky, see Creating Time of Day Sky Effects (p. 1269).

Topics

- Environment Lighting (p. 1779)
- Environment Shadows (p. 1785)

Environment Lighting

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard uses physically-based lighting and shading models to implement global illumination and environment lighting.

For information about the Light entity and the Environment Probe entity used in environment lighting, see Light Entities.

For information about using the Time of Day Editor to simulate the changing lighting effects caused by the sun moving across the sky, see Creating Time of Day Sky Effects (p. 1269).

Topics

- Illuminance and Auto Exposure Key (p. 1780)
Illuminance and Auto Exposure Key

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Also known as luminous flux density, illuminance is the total amount of visible light falling on a point on a surface from all directions above the surface in a given time. Proper illuminance values ensure the environment lighting in your level closely models real-world values. Besides simply having good ratios between light and dark, accurate illuminance values ensure that tone-mapping, and eye adaptation works optimally.

The following table lists real-world illuminance values, expressed in luminous flux (lux). Lux is the unit of illuminance and luminous emittance, measuring lux per unit area, and equal to one lumen per square meter.

<table>
<thead>
<tr>
<th>Real-world illuminance</th>
<th>Lux Value</th>
<th>Uniformity Ratio</th>
<th>Artistic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full moon</td>
<td>0.25</td>
<td>0.00005</td>
<td>-</td>
</tr>
<tr>
<td>Living room</td>
<td>50</td>
<td>0.01</td>
<td>-</td>
</tr>
<tr>
<td>Clear sunrise</td>
<td>400</td>
<td>0.08</td>
<td>-</td>
</tr>
<tr>
<td>Office</td>
<td>500</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>TV studio</td>
<td>1,000</td>
<td>0.2</td>
<td>-</td>
</tr>
<tr>
<td>Overcast day</td>
<td>15,000</td>
<td>3.0</td>
<td>~ 1.5</td>
</tr>
<tr>
<td>Indirect sunlight (in shadow)</td>
<td>20,000</td>
<td>4.0</td>
<td>~ 2.0</td>
</tr>
<tr>
<td>Direct sunlight</td>
<td>100,000</td>
<td>20.0</td>
<td>~ 10.0</td>
</tr>
</tbody>
</table>

The Auto Exposure Key setting controls the amount of light exposure and determines whether the tone-mapped image appears relatively bright or dark. This setting is calculated automatically from the average scene illuminance, which is why it is important to use standard real-world illuminance levels. For other settings that affect the tone mapping of a scene, see HDR Settings (p. 1781).

Lumberyard's auto-exposure mode works in exposure value (EV) units and can be enabled using the r_HDREyeAdaptationMode console variable.

The following settings are used to achieve the desired illuminance in an environment level. See Setting Daytime Atmospheric Effects (p. 1266) for more information.

- Sun color
- Sun color multiplier
- Sun intensity
• Sun intensity multiplier

**HDR Settings**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

As discussed in Illuminance and Auto Exposure Key (p. 1780), the auto exposure key setting controls the amount of scene exposure and determines whether the tone-mapped image appears relatively bright or dark. Several other settings also affect the tone mapping of scene. These are known collectively as HDR (high dynamic range) in the **Time of Day Editor**.

Film curve parameters in the **Time of Day Editor** correspond to analogous parameters that exist for camera film. A film curve has three distinct regions with different contrast transfer characteristics:

- The lower part of a film curve that is associated with relatively low exposures is designated the toe, and corresponds to the low-density portions of an image. When an image is exposed so that areas fall within the toe region, little or no contrast is transferred to the image.
- The upper part of a film curve that is associated with relatively high exposures is designated the shoulder, and corresponds to the high-density portions of an image. When an image is exposed so that areas fall within the shoulder region, little or no contrast is transferred to the image.
- The middle part of a film curve with the highest level of contrast is produced within a range of exposures falling between the toe and the shoulder, and is designated the midtones region. This portion of the curve is characterized by a relatively straight and steep slope in comparison to the toe and shoulder regions. You should adjust your image so that important areas fall within this region for maximum contrast.

**To set HDR settings parameters**

1. In Lumberyard Editor, choose **Tools, Other, Time Of Day**.
2. Choose **HDR Settings**, and for **HDR**, adjust the values of the following settings.

<table>
<thead>
<tr>
<th>HDR Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Film curve shoulder scale</td>
<td>Slope at the tip of the HDR curve (modifies bright values).</td>
</tr>
<tr>
<td>Film curve midtones scale</td>
<td>Linearity of the middle of the HDR curve (modifies gray values).</td>
</tr>
<tr>
<td>Film curve toe scale</td>
<td>Slope at the base of the curve (modifies dark values).</td>
</tr>
<tr>
<td>Film curve whitepoint</td>
<td>Value to be mapped as pure white or reference white in the tone-mapped image.</td>
</tr>
<tr>
<td>Saturation</td>
<td>Color saturation before tone-mapping.</td>
</tr>
<tr>
<td>Color balance</td>
<td>Overall color of the scene.</td>
</tr>
<tr>
<td>Auto Exposure Key</td>
<td>Overall brightness of the scene used for eye adaptation. Eye adaptation causes the exposure of a scene to simulate the way human eyes adjust when going from a brightly lit</td>
</tr>
</tbody>
</table>
### HDR Settings

<table>
<thead>
<tr>
<th>HDR Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>environment to a dark environment and vice versa. Use lower value for dark scenes and higher values for bright scenes.</td>
</tr>
<tr>
<td></td>
<td>Default value is 0.18.</td>
</tr>
<tr>
<td><strong>Auto Exposure Min</strong></td>
<td>Darkest possible exposure used for eye adaptation.</td>
</tr>
<tr>
<td><strong>Auto Exposure Max</strong></td>
<td>Brightest possible exposure used for eye adaptation.</td>
</tr>
<tr>
<td><strong>Bloom amount</strong></td>
<td>Controls the amount of bloom that comes from glowing or lit objects.</td>
</tr>
</tbody>
</table>


## Global Environment Lighting

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To implement global lighting for an entire level, you use a global environment probe (also known as a global light probe) and associated generated cubemap.

Environment probes control many aspects of the physically based lighting in Lumberyard, including accurate shadow colors, ambient diffuse values, and specular reflections. They also provide bounce lighting by taking the colors from the surroundings and applying them directly to the diffuse color of materials inside their radius.

When placing environment probes in a level, pay attention to how probes are layered and sorted going from global to local probes.

Every level should have a global environment probe. Global probes provide the entire level with ambient lighting, which is calculated from the probe's location. In addition to a global probe, a level may have one or more local probes. For more information about local probes, see Local Environment Lighting (p. 1784).

As shown in the following table, the probe has several configurable properties, which you can adjust in the Rollup Bar.

### EnvironmentProbe Properties

**Active**

Enables and disables the probe.
**BoxSizeX, BoxSizeY, BoxSizeZ**

Specifies the XYZ dimensions of the probe's area of effect. Probes are projected as cubes in the level. For a global probe, set values large enough to span the entire level.

**Diffuse**

Sets the diffuse color of the light. Set to 255, 255, 255.

**DiffuseMultiplier**

Makes the light brighter. Set to 1.

**SpecularMultiplier**

Multiplies the specular color brightness. Set to 1.

**AffectsThisAreaOnly**

Set parameter to False to make lights cover other vis areas.

**AttenuationFalloffMax**

Controls the falloff amount (0–1) to create smoother transitions or hard edges. A value of 0.8 means that falloff begins at 80% at the boundaries of the box. Set value to 0 for a global probe (no falloff).

**IgnoresVisAreas**

Controls whether the light should respond to vis areas. Set value to True for a global probe.

**SortPriority**

Gives control over which probe has more visual interest and therefore a higher priority. Set the value to 0 for a global probe, then increase the value for local probes, where higher values indicate more localized probes.

**deferred_cubemap**

Specifies the file location of the cubemap texture.

**BoxHeight**

Adjusts the height of cubemap box.

**BoxLength**

Adjusts the length of cubemap box.

**BoxProject**

When enabled, Lumberyard factors in the size of the cubemap box.

**BoxWidth**

Adjusts the width of cubemap box.

---

**To generate a global cubemap**

1. In Rollup Bar, under Objects, click Misc, EnvironmentProbe.
2. Click to place the probe in your level.
3. Under EnvironmentProbe Params, leave cubemap_resolution at the default 256. This is the optimal resolution for best performance.
4. Select the preview_cubemap check box to see the cubemap in your level.
5. Under EnvironmentProbe Properties, adjust the following property values to configure the probe to be global:
   - **BoxSizeX, BoxSizeY, and BoxSizeZ** values: Large enough to span the entire level
   - **Diffuse** color value: 255, 255, 255
• **DiffuseMultiplier** and **SpecularMultiplier** values: 1
• **SortPriority**: 0
• **AttenuationFalloffMax**: 0
• **IgnoreVisAreas**: True (check box selected)

6. Click **Generate Cubemap**. Lumberyard creates three textures in `textures\cubemaps\your_level`— one for the diffuse map, one for the specular map, and one for the source .t.i.f file.

7. To check your cubemap for accuracy, create and then place a smooth, reflective sphere entity near the probe. If its surface looks different from the environment around it, you need to regenerate the cubemap.

8. Click **Generate Cubemap** again. This incorporates object reflections from the originally generated cubemap for added realism.

9. To hide the sphere entity in your level, select its **HiddenInGame** check box, found under **Entity Params** in the **Rollup Bar**.

### Local Environment Lighting

This topic references tools and features that are **legacy**. If you want to use legacy tools in Lumberyard Editor, disable the **CryEntity Removal gem** using the **Project Configurator** or the **command line**. To learn more about legacy features, see the **Lumberyard Legacy Reference**.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the AWS Game Tech blog to learn more.

Lumberyard uses local environment probes and their generated cubemaps to implement local lighting. The purpose of local cubemaps is to light smaller areas more accurately. This ensures that all areas in your level have accurate lighting effects that may not be covered by the global cubemap. Lumberyard automatically gives a local probe higher priority within its defined radius and superimposes its effects on those of the global probe. For more information about global probes, see **Global Environment Lighting** (p. 1782).

When placing environment probes in a level, pay attention to how probes are layered and sorted going from global to local probes.

**To generate a local cubemap**

1. In the **Rollup Bar**, under **Objects**, click **Misc**, **EnvironmentProbe**.
2. Click to place in the probe in your level.
3. Under **EnvironmentProbe Params**, leave the **cubemap_resolution** at 256, the default. This is the optimal resolution for performance.
4. Select the **preview_cubemap** check box to see the cubemap in your level.
5. Under **EnvironmentProbe Params** and under **EnvironmentProbe Properties**, adjust property values for the desired effect. For more information about these properties, see the table in **Global Environment Lighting** (p. 1782).
6. Click **Generate Cubemap**.

Lumberyard creates three textures in `textures\cubemaps\your_level`— one for the diffuse map, one for the specular map, and one for the source .t.i.f file.

7. To check your cubemap for accuracy, create and then place a smooth, reflective sphere entity near the probe. If its surface looks different from the environment around it, you need to regenerate the cubemap.
8. Click **Generate Cubemap** again. This incorporates object reflections from the originally generated cubemap for added realism.

9. To hide the sphere entity in your level, select its **HiddenInGame** check box, found under **Entity Params** in the **Rollup Bar**.

---

### Environment Shadows

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

---

Lumberyard supports shadow casting from all light sources and shadow receiving on all deferred and most forward-rendered geometry. Traditional shadow mapping is used for shadow generation. Light sources can be directional, such as from the sun and moon, or from point and area light sources.

As shadow generation is resource-intensive, Lumberyard offers the following features to mitigate this:

- You can control the degree to which Lumberyard caches shadows and stops dynamically updating the most distant cascaded sun shadows.
- You can set point and area light sources to be updated in intervals, such as every second frame.
- You can use the `r_MergeShadowDrawcalls` console variable to merge submaterials during shadow generation, resulting in fewer drawcalls.

**Topics**

- Cached Shadows (p. 1785)
- Object Shadows (p. 1786)
- Shadow Proxies (p. 1787)
- Using CVARs to Configure Terrain and Cascade Shadow Maps (p. 1787)

### Cached Shadows

Shadow caching is an effective optimization method to reduce the number of shadow drawcalls and to increase the shadow casting and receiving range.

Starting from a defined cascade number, Lumberyard can render subsequent shadow cascades and then keep them in memory. Once the cached cascade is initialized, no more draw calls are needed for updates. This enables long-range distant shadows with almost no performance cost.

Keep in mind that cached shadows are memory intensive, with the default configuration requiring approximately 130 MB of video memory.

In addition, ensure that all shaders are compiled before triggering an update or all objects may not be rendered into the cached shadow maps.

**Placement and Update**

Cached shadow cascades are centered around the rendering camera by default, and automatically recenter and update once the camera gets close to the cascade border.
You can override this automated placement by using the **Recompute Static Shadows** Script Canvas node, which takes the world space Min and Max input positions of the bounding area for the first cached cascade. Bounding boxes for subsequent cached cascades are scaled versions of the preceding cascades and are based on the **NextCascadeScale** input multiplier. The **Trigger** input causes an update of all cached shadow cascades.

**Note**
To keep you informed, a warning message appears in the console each time a cached shadow cascade is updated.

### Dynamic Distance Shadows

Cached shadows work well with static objects, but dynamic objects don't get their shadows updated while moving. To overcome this, you can selectively exclude dynamic objects from the cache and render them to the standard cascades. The performance overhead of enabling this feature for a limited number of entities is generally low.

**To enable dynamic distance shadows for an object**

- Select the **DynamicDistanceShadows** check box for the entity.

### Console Variables

When Lumberyard is set to place shadows automatically, the selected resolution combined with the desired world space pixel density, which is derived from the approximate logarithmic split scheme, determines the world space area covered by each shadow cascade. Lowering the resolution lowers the shadowed range for each cascade while still maintaining shadow quality.

When you place shadows manually, the resolution is uniformly stretched across the shadow cascade. Consequently, lower resolutions result in lower shadow quality at the same world space coverage.

Use the following console variables to control cached shadows, including setting the placement and resolution for individually cached shadow cascades.

- **r_ShadowsCache** – Caches all sun shadow cascades above the value. 0 = no cached cascades, 1 = cache first cascade and up, 2 = cache second cascade and up.
- **r_ShadowsCacheResolutions** – The resolution of the cached cascades.
- **r_ShadowsCacheFormat** – Storage format for cached shadow maps: 0 = D32: 32 bit float, 1 = D16: 16 bit integer.
- **e_ShadowsCacheUpdate** – Triggers updates of cached shadow maps: 0 = no update, 1 = one update, 2 = continuous updates.
- **e_ShadowsCacheObjectLod** – The level of detail (LOD) used for rendering objects into the cached shadow maps.
- **e_ShadowsCascadesDebug** – Enables debug view mode. 0 = disable, 1 = enable.
- **e_DynamicDistanceShadows** – Toggles support for having selected objects cast dynamic shadows.

### Object Shadows

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With object shadows, you can assign custom shadow maps to selected objects, resulting in increased shadow quality due to higher world space shadow texel (texture element) density and reduced depth range.
The drawbacks of using object shadows are increased memory consumption of the additional shadow maps and increased shadow filtering cost.

Object shadows only affect sun shadows. For performance reasons they are not sampled on forward geometry such as particles, hair, and eyes.

**Using I3DEngine**

The following I3DEngine interface functions can be called from anywhere in game code.

- **AddPerObjectShadow** – Adds an object shadow.
- **RemovePerObjectShadow** – Removes an object shadow.
- **GetPerObjectShadow** – Retrieves object shadow settings for a given RenderNode. Do not overwrite the RenderNode pointer. Instead use `AddPerObjectShadow\RemovePerObjectShadow`.
- **ShadowMapSize**: Size of the custom shadow map, which is automatically rounded to the next power of two.

**Console Variables**

You can use the `e_ShadowsPerObject` console variable with object shadows. With this variable, 0 = off, 1 = on, and -1 = don't draw object shadows.

**Shadow Proxies**

Shadow proxies are a method of significantly reducing shadow performance costs by creating dedicated low-polygon count geometry to cast an object's shadow with minimal visual differences. You can also use shadow proxies to minimize shadow artifacts by controlling which geometry can cast shadows.

Keep in mind that if the shadow proxy mesh aligns closely with the RenderMesh, you may notice self-shadow artifacts.

No material setup is required in your DCC tool. Instead you use the Material Editor to set up shadow proxies in the material using Material Editor. Place the shadow proxy on its own submaterial, setting **Opacity** to 0 and ensuring that **No Shadow** is not selected (the default).

The shadow proxy must also be linked as a child node of the RenderMesh, and it must be on its own material ID.

For the RenderMesh material, set as you normally would, except under the **Advanced** properties, select the **No Shadow** option. This instructs Lumberyard to use the shadow proxy instead of the RenderMesh to render the shadows.

**Using CVARs to Configure Terrain and Cascade Shadow Maps**

You can use console variables (CVARs) in Amazon Lumberyard to configure terrain and cascade shadow maps (also known as gradient shadow maps, or GSMs). This topic provides a list of key shadow-related console variables and shows you how to save your changes to them. It also shows you how to learn about other related console variables.
Topics

- Some Key Shadow-Related Console Variables (p. 1788)
- Saving Console Variable Changes (p. 1794)
- Finding Other Shadow-Related Console Variables (p. 1794)
- Tutorial: Using Console Variables to Tune Cascade Shadows (p. 1795)

Some Key Shadow-Related Console Variables

The following table shows some of the console variables that configure shadows, gradient shadow maps, and related debug modes.

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_GsmCastFromTerrain</td>
<td>int</td>
<td>Specifies that shadows are cast from terrain. Off by default.</td>
</tr>
<tr>
<td>e_GsmDepthBoundsDebug</td>
<td>int</td>
<td>Shows bounding volumes related to gradient shadow maps.</td>
</tr>
<tr>
<td>e_GsmLodsNum</td>
<td>int</td>
<td>Specifies the number of gradient shadow map levels of detail (LOD).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 to 7.</td>
</tr>
<tr>
<td>e_GsmRange</td>
<td>float</td>
<td>Size of the LOD 0 gradient shadow map area in meters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 3 meters</td>
</tr>
<tr>
<td>e_GsmRangeStep</td>
<td>float</td>
<td>Specifies a step value to multiply the previous range by to determine the range of the next GSM LOD.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To determine the step size, use the following guidelines:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If e_GsmRange is 3, cascade 0 is 3m.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If e_GsmRangeStep is 3, the following values result.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cascade 1 is 3m x 3m = 9m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cascade 2 is 9m x 3m = 27m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cascade 3 is 27m x 3m = 81m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cascade 4 is 81m x 3m = 243m</td>
</tr>
<tr>
<td>e_GsmStats</td>
<td>int</td>
<td>When enabled, shows debug information about shadow cascades.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td>e_ObjShadowCastSpec</td>
<td>int</td>
<td>Specifies that objects that have a shadow cast spec less than or equal to the specified value cast shadows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 4</td>
</tr>
</tbody>
</table>
## Environment Shadows

### Console Variable

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_ParticleShadowsNumGSMs</td>
<td>int</td>
<td>Specifies the number of shadow GMSs that are used for particle shadows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>e_ParticlesShadows</td>
<td>int</td>
<td>Enables shadows on particles.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Force</td>
</tr>
<tr>
<td>e_Shadows</td>
<td>int</td>
<td>Activates the drawing of shadows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>e_ShadowsBlendCascades</td>
<td>int</td>
<td>Softens the blend between shadow cascade regions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>e_ShadowsBlendCascadesVal</td>
<td>float</td>
<td>Specifies the width of the region across which two cascades blend.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0.75</td>
</tr>
<tr>
<td>e_ShadowsCascadesDebug</td>
<td>int</td>
<td>For debugging purposes, draws each shadow cascade with a different color on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>screen.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td>e_ShadowsCastViewDistRatio</td>
<td>float</td>
<td>Specifies the view distance ratio for shadow maps that cast from objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>e_ShadowsClouds</td>
<td>int</td>
<td>Enables shadows for clouds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>e_ShadowsConstBias</td>
<td>float</td>
<td>Specifies the shadow slope bias for shadow generation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The shadow bias is the distance from where the shadows are cast to where they</td>
</tr>
<tr>
<td></td>
<td></td>
<td>can be received.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Console Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>e_ShadowsConstBiasHQ</td>
<td>float</td>
<td>Specifies the shadow slope bias for shadow generation in high quality mode. When shadow mode is set to high quality, the value of <code>e_ShadowsConstBiasHQ</code> or <code>e_ShadowsConstBias</code> is used, whichever is smaller. Default value: 0.05</td>
</tr>
<tr>
<td>e_ShadowsDebug</td>
<td>int</td>
<td>Displays a debug viewport for shadow maps.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td>e_ShadowsLodBiasFixed</td>
<td>int</td>
<td>Specifies the number of LOD levels that simplify mesh for shadow map generation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>e_ShadowsMaxTexRes</td>
<td>int</td>
<td>Sets the maximum resolution of the shadow map. Typical values are 256 for faster performance, 512 for a balance of performance and quality, and 1024 for higher quality. Default value: 1024</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Larger values can cause texture thrashing warnings. To compensate, increase the value of <code>r_DynTexMaxSize</code>.</td>
</tr>
<tr>
<td>e_ShadowsPoolSize</td>
<td>int</td>
<td>Uses the formula <code>e_ShadowsPoolSize * e_ShadowsPoolSize</code> to set the size of the shadow pool. The shadow pool is the texture atlas used for rendering shadows from individual light points. Depending on the quality required, each light uses a different sized portion of the shadow map.</td>
</tr>
<tr>
<td>e_ShadowsResScale</td>
<td>float</td>
<td>Specifies the resolution scale for an individual shadow inside the shadow pool for use by point lights. Determines how much of the shadow pool atlas to use when rendering the shadows for a light entity. Default value: 4</td>
</tr>
<tr>
<td>e_ShadowsSlopeBias</td>
<td>float</td>
<td>Specifies the shadow slope bias. The shadow slope bias is the distance from where shadows are cast to where they can be received, scaled by the slope of the geometry. Default value: 1</td>
</tr>
<tr>
<td>Console Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>e_ShadowsSlopeBiasHQ</td>
<td>float</td>
<td>Specifies the shadow slope bias in high quality mode. When shadow mode is set to high quality, the value of e_ShadowsSlopeBiasHQ or e_ShadowsSlopeBias is used, whichever is smaller. Default value: 0.25</td>
</tr>
<tr>
<td>e_ShadowsTessellateCascades</td>
<td>int</td>
<td>Specifies the maximum cascade number to render tessellated shadows. In cascades higher than the specified value, objects render shadows without tessellation. Default value: 1 Note Use this console variable to optimize performance.</td>
</tr>
<tr>
<td>e_ShadowsTessellateDLights</td>
<td>int</td>
<td>Enables or disables tessellation for shadows cast by local lights. 0 = Off (default) 1 = On</td>
</tr>
<tr>
<td>e_ShadowsUpdateViewDistRatio</td>
<td>int</td>
<td>Specifies the view distance ratio for shadow maps that update for the shadow pool. Default: 256</td>
</tr>
<tr>
<td>e_ShadowsOnAlphaBlend</td>
<td>int</td>
<td>Enables shadows on alpha blend. 0 = Off (default) 1 = On</td>
</tr>
<tr>
<td>r_DrawNearShadows</td>
<td>int</td>
<td>Enables shadows for near objects. 0 = Off 1 = On (default)</td>
</tr>
<tr>
<td>r_DynTexMaxSize</td>
<td>int</td>
<td>Specifies the maximum dynamic texture size. Use in conjunction with e_ShadowsMaxTexRes. Default value: 80</td>
</tr>
<tr>
<td>r_FogShadows</td>
<td>int</td>
<td>Enables deferred volumetric fog shadows. 0 = Off 1 = Standard resolution (default) 2 = Reduced resolution</td>
</tr>
<tr>
<td>Console Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>r_fogShadowsWater</code></td>
<td>int</td>
<td>Enables volumetric fog shadows for water volumes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td><code>r_ShadowJittering</code></td>
<td>float</td>
<td>Specifies the shadow map jittering radius. Values set by this CVAR are overwritten by ToD animation as soon as the ToD changes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 2.5</td>
</tr>
<tr>
<td><code>r_ShadowPoolMaxFrames</code></td>
<td>int</td>
<td>Specifies the maximum number of frames during which a shadow can exist in the shadow pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><code>r_ShadowPoolMaxTimeslicedUpdatesPerFrame</code></td>
<td>int</td>
<td>Specifies the maximum number of time-sliced shadow pool updates that are allowed per frame.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><code>r_ShadowsCache</code></td>
<td>int</td>
<td>Replaces all sun cascades above the specified value with a cached (static) shadow map.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = No cached shadows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Replace the first and following cascades.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Replace the second and following cascades.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[...]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td><code>r_ShadowsCacheFormat</code></td>
<td>int</td>
<td>Specifies the texture format for shadow cache.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Use D32 texture format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Use D16 texture format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td><code>r_ShadowsCacheResolutions</code></td>
<td>string</td>
<td>Specifies the resolution of the shadow cache per cascade.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 4214</td>
</tr>
<tr>
<td><code>r_ShadowsPCFFiltering</code></td>
<td>int</td>
<td>Specifies whether to use Percentage Closer Filtering (PCF) for shadows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
</tbody>
</table>
## Environment Shadows

### Console Variables

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_ShadowsScreenSpace</td>
<td>int</td>
<td>Specifies whether to include screen space tracing in shadow computations. Screen space tracing helps to reduce artifacts caused by limited shadow map resolution and biasing. Use in cutscenes for better shadows on character faces.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Off (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td>Note</td>
<td></td>
<td>This effect is applied only in the near range.</td>
</tr>
<tr>
<td>r_ShadowsUseClipVolume</td>
<td>int</td>
<td>Specifies whether shadows use clip volume.</td>
</tr>
<tr>
<td>DUMPTODISK</td>
<td></td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>sys_spec_shadows</td>
<td>int</td>
<td>A console variable group that applies settings to certain shadow variables to configure shadow quality. A higher value signifies higher quality.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 1 to 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For information on the console variables included in this group and how their settings are affected, see sys_spec_shadows Console Variables (p. 1793).</td>
</tr>
</tbody>
</table>

### sys_spec_shadows Console Variables

The following table shows the console variables in the `sys_spec_shadows` group. The columns show the value of each variable for the `sys_spec_shadows` settings of 1 through 8. The final column lists the default values.

<table>
<thead>
<tr>
<th>sys_spec_shadows</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_GsmLodsNum</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>e_GsmRange</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>e_ObjShadowCastSpec</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>e_ParticlesShadows</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_Shadows</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_ShadowsBlendCascades</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_ShadowsCastViewDistRatio</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_ShadowsClouds</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_ShadowsLodBiasFixed</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e_ShadowsMaxTexRes</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>e_ShadowsOnAlphaBlend</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Saving Console Variable Changes

Changes that you make to CVARs within the console do not automatically save in your level or editing session. To save your changes, you can use configuration files or Lua script.

**To save console variable changes**

- Do one of the following:
  - Edit and save one of the console variable configuration files. For more information, see [Configuring Console Variables](#).
  - In a Lua script, use the following syntax to assign a value to the variable at startup:
    ```lua
    ConsoleRequestBus.Broadcast.ExecuteConsoleCommand('r_ConsoleVariable = Value')
    ```

### Finding Other Shadow-Related Console Variables

You can use the Lumberyard Editor console window to search for additional shadow-related console variables.

**To find additional shadow-related console variables**

1. In Lumberyard Editor, choose **Tools, Console**.
2. In the console window, do one of the following to open the **Console Variables** search window.
A. Click the X icon next to the command entry area.
B. Right-click the command entry area.

3. In the **Console Variables** window, for **Search**, enter **shadow** or **gsm**.

4. To learn more about a console variable, pause your pointer on the variable name.

For more information about using the Lumberyard Editor console, see **Using the Console Window** (p. 210).

**Tutorial: Using Console Variables to Tune Cascade Shadows**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This topic shows you how to use console variables (CVARs) to tune the shadows in a scene. The sample is a 2k island test level that was generated with World Machine.

**Topics**
- Prerequisite: Disable Rendering Features (p. 1796)
• Step 1: Enable Cascade Shadows (p. 1796)
• Step 2: Enable Cascades Debug Mode (p. 1798)
• Step 3: Maximize the Number of Cascades (p. 1800)
• Step 4: Change the Size of the Root Cascade (p. 1802)
• Step 5: Change the Step Size (p. 1806)
• Step 6: Enable Shadows Debug Mode (p. 1809)

Prerequisite: Disable Rendering Features

Before you begin, disable rendering features like fog. Rendering features like fog can obscure the view of shadows at a distance. To disable fog, enter the following command:

```
e_Fog=0
```

Step 1: Enable Cascade Shadows

By default, shadow cascades are disabled for terrain, as the following image shows.
To enable cascade shadows from terrain, enter the following command:

```
e_GsmCastFromTerrain=1
```

The left side of the cliff in the foreground now has shadows.
To disable cascade shadows, set `e_GsmCastFromTerrain` to 0.

\[
e_{\text{GsmCastFromTerrain}} = 0
\]

**Step 2: Enable Cascades Debug Mode**

You can use debug mode to better see the cascades and the effects of shadow-related console variables. To enable debug mode, enter the following command:
In debug mode, each cascade is assigned a separate color. In the following example, the nearest cascade is in red and the farthest in pink.

| Cascade 0: Red | Cascade 1: Green | Cascade 2: Blue | Cascade 3: Yellow | Cascade 4: Pink |

This information is displayed in the viewport, as the following image shows. Areas that do not have these special colorings do not have shadows.
Step 3: Maximize the Number of Cascades

To create the highest quality shadows, start by maximizing the number of cascades and then scale back as needed. To maximize the number of cascades, enter the `e_GsmLodsNum` console variable, which specifies the number of gradient shadow map levels of detail (LOD). The default is 5.

**Note**
The default of 5 is true even when the `sys_spec_shadows` group console variable is set to its highest value (8).
To increase the number of cascades to 7, enter the following command:

```
e_GsmLodsNum=7
```

In the example, this command expands the set of debug cascade colors to the following:

- Cascade 0: Red
- Cascade 1: Green
- Cascade 2: Blue
- Cascade 3: Yellow
- Cascade 4: Pink
- Cascade 5: Light Blue
- Cascade 6: Red Orange

**Note**

The additional cascade numbers and corresponding colors are not listed in the upper left of the viewport but do appear on the terrain.

The light blue color shows that the shadow range has been extended much further into the distance.
Step 4: Change the Size of the Root Cascade

In a scene, the terrain scales away from the root cascade 0 in the foreground to higher numbered cascades in the background. To specify the size of the root cascade 0 in meters, use the e_GsmRange console variable. The examples in this section show how e_GsmRange values affect shadows.

The first example sets e_GsmRange to 0.5.

```
e_GsmRange=0.5
```
The resulting image shows that Cascade 0 (red) is not yet visible. Cascades 1 through 6 (green, blue, yellow, pink, light blue, and red-orange) are successively visible the farther one looks into the distance. Beyond the last cascade, the mountain has its usual brownish color.

The following command increases the range of the root cascade to 1 meter.

```
e_GsmRange=1.0
```

At this setting, Cascade 0 is visible in the foreground in red. The range has increased to include the distant mountain in Cascade 6 (red-orange).
The following command changes the range of the root cascade to 3 meters, which is the default.

```
e_GsmRange=3.0
```

In the example, this setting expands the area of the red Cascade 0 in the foreground. The other cascades are pushed farther back. The mountain in the distance, which was Cascade 6 (red-orange) is now Cascade 5 (light blue), and Cascade 6 is no longer visible.
At this point, you can reduce the number of cascades from 7 to 6 with the following command.

```
e_GsmLodsNum=6
```

As the following image shows, the command has no visual impact on the range of shadows.
Step 5: Change the Step Size

The range step size specifies how much the next cascade changes. The following command specifies a range of 3, which is the default.

```
e_GsmRangeStep=3
```

The following command changes the range to 1.5.
The following image shows how this significantly reduces the range. In the example, the number of cascades is still 6 (\texttt{e\_GsmLodsNum=6}) and the root cascade is still at 3 meters (\texttt{e\_GsmRange=3.0}). However, the mountains in the distance no longer have shadows.

The next example maintains the same settings but changes the range step to 2.7.

This scales the cascades out again so that they fall just short of the furthest mountain in the distance.
The following example lowers the number of cascades from 6 to 5 but increases the range step to 3.5.

```plaintext
e_GsmLodsNum=5
e_GsmRange=3.0
e_GsmRangeStep=3.5
```

Because of the increase in range step, the reduced number of cascades covers approximately the same amount of terrain as in the previous example. The area which was previously Cascade 5 (light blue) is now Cascade 4 (pink).
Step 6: Enable Shadows Debug Mode

The `e_ShadowsDebug` debug mode, unlike the `e_ShadowsCascadesDebug` console variable presented earlier, shows you information about each cascade in individual windows in the viewport. To enable shadows debug mode, enter the following command:

```
e_ShadowsDebug=1
```

The following image shows the viewport with shadows debug mode enabled.
The following example in shadows debug mode uses the maximum number of cascades (7) but adjusts their reach by tuning the range step to 2.25.

```plaintext
e_ShadowsDebug=1
e_GsmLodsNum=7
e_GsmRange=3.0
e_GsmRangeStep=2.25
```
Create screen effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard’s rendering technology starts with a modern, physically-based shading core that renders materials based on real world physical parameters (such as base color, metallicity, smoothness, and specularity), allowing you to achieve realistic results using the same physically-based parameters used in the highest end film rendering pipelines.

The rendering core is supplemented by a rich set of the most frequently used real time lighting, shading, special effects, and post effects features, such as physical lights, global illumination, volumetric fog, procedural weathering, particle systems, dynamic real time shadows, motion blur, bokeh depth of field, post color correction, and more.
Lumberyard's rendering engine is tightly integrated with Lumberyard Editor, so the graphical fidelity and performance achieved in your game is what you see in the editor. Changes made in the editor are instantly reflected in the fully rendered scene, allowing for immediate feedback and rapid iteration.

The Lumberyard rendering technology is designed to take maximum advantage of today's high-end PC and console operating systems, while maintaining compatibility with older hardware by scaling down graphical features and fidelity without compromising the core visual elements of your scene.

Topics
- Working with camera screen effects (p. 1812)
- Temporal Antialiasing and Supersampling (p. 1815)
- Voxel-based Global Illumination (SVOGI) (p. 1818)
- Fog Systems (p. 1822)
- Render Nodes (p. 1829)
- Generating Stars .dat File (p. 1833)
- Building DirectX 12 Supported Applications (p. 1834)

Working with camera screen effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use rendering cameras to define custom views within your level. You can trigger them using the Track View or the Enable Depth Of Field Script Canvas node. Rendering cameras are used frequently for animated sequences.

For more information about how to add and use rendering cameras, see Camera Entity.

Depth of Field

Lumberyard uses an efficient gather-based depth of field (DOF) implementation. Depth of field is used to enhance the realism of a scene by simulating the way a real-world camera works. Use a broad depth of field to focus on all or nearly all of a scene. Use a narrow depth of field to focus on objects that are within a certain distance from the camera.
You can enable depth of field by using the `r_depthOfFieldMode` console variable. To control depth of field use the Track View or the `Enable Depth Of Field` Script Canvas node.

**Motion Blur**

Lumberyard uses a sample-weighted motion blur implementation whose settings mirror real-world camera shutter speed settings.
Temporal Antialiasing and Supersampling

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Supersampling is a method of antialiasing that attempts to reduce jagged, pixelated edges (aliasing) in images. It works by sampling a higher resolution version of the image to get the average color of a pixel before reducing it to the intended size. The averaged color values create smoother transitions along an edge of colors, reducing the jagged look.

Because supersampling is memory intensive, Lumberyard uses temporal antialiasing (TAA) to approximate supersampling. While supersampling analyzes pixels spatially, TAA analyzes frames over time, even when the camera is static. The current frame is projected onto the previous frame and samples are blended into an accumulation buffer. This technique reduces ghosting artifacts in motion and gives you control over the amount of antialiasing needed for your graphics. You can create sharp images or softer, blurred images. You can also use supersampling for very high quality rendering.

TAA is useful for reducing aliasing from the following:

- Specular highlights and bright pixels, especially when combined with convolution filters such as Depth of Field or Bloom.
- Geometric and alpha-tested edges.

You can configure TAA by setting the `r_AntialiasingMode` console variable, editing the platform configuration file, or creating a level configuration file with this setting.

Controlling Antialiasing

The following table lists the antialiasing modes that are available in Lumberyard when you use the `r_AntialiasingMode` console variable.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Console Variable Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No antialiasing</td>
<td>0</td>
<td>Disables postprocessing-based antialiasing. This is useful for debugging or when the technique is unnecessary. You may also use a higher resolution if you prefer not to spend system resources on antialiasing.</td>
</tr>
<tr>
<td>FXAA</td>
<td>1</td>
<td>Enables fast approximate antialiasing (FXAA), which filters edges using a postprocessing edge detection technique.</td>
</tr>
<tr>
<td>SMAA (1TX)</td>
<td>2</td>
<td>Enables subpixel morphological antialiasing (SMAA), which uses an advanced postprocessing technique to detect edges in order to filter edges. This mode includes a basic temporal antialiasing component but does not address subpixel jitter.</td>
</tr>
<tr>
<td>TAA</td>
<td>3</td>
<td>Enables temporal antialiasing (TAA). This is the default mode.</td>
</tr>
</tbody>
</table>

The following table lists the additional console variables that you can configure for temporal antialiasing.

Version 1.28
1815
<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| r_AntialiasingModeEditor             | Specifies whether to use subpixel jitter in the editor. This can eliminate jiggling artifacts on helper objects at the expense of losing antialiasing on static scenes.  
Valid values: 0 = disables this mode | 1 = enables this mode                                                                                                                                    |
| r_AntialiasingTAAClampingFactor      | Controls the clamping factor in standard deviations. Set the value at which to limit the TAA response. Higher values create a more stable scene with less flickering but more ghosting.  
Lower values create a scene with less ghosting but more aliasing and flickering.  
Valid values: 0.75 – 2.0                                                                                     |
| r_AntialiasingTAAJitterPattern       | Specifies the sampling pattern for temporal antialiasing.                                                                                                                                                     |
|                                      | Valid values: 0 = no subsamples | 1 = 2x | 2 = 3x | 3 = 4x | 4 = 8x | 5 = sparse grid, 8x8 | 6 = random | 7 = Halton 8x (default) | 8 = Halton random |
| r_AntialiasingTAA gestionMax         | Clamps the input luminance before temporal filtering to help with image stability. Extra bright pixels can ghost and cause bloom artifacts.                                                               |
|                                      | Default value: 100.0                                                                                                                               |
| r_AntialiasingTAA MotionDifferenceMax | Specifies the maximum difference of speed between the current pixel and its history pixel at which the current pixel is considered fully disoccluded. Lower values create a scene with less ghosting but more aliasing on disoccluded pixels. |
| r_AntialiasingTAA MotionDifferenceMaxWeight | Specifies the blend weight for the current frame at the maximum speed difference (defined by the r_AntialiasingTAA MotionDifferenceMax console variable).                     |
|                                      | Valid values: 0 = 100% of the history pixel | 1 = 100% of the current pixel                                                                                                                                |
|                                      | Default value: 0.5                                                                                                                               |
| r_AntialiasingTANewFrameFaloff       | Represents the amount of time, in seconds, for the history signal to reach 63% of the source signal. Lower values create faster convergence, which can reduce ghosting but introduce some aliasing. Higher values create slower convergence.  
Because higher values may not improve stability, you should adjust the color clamping factor before adjusting this value.  
Default value: 0.15                                                                                                                                 |
<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_AntialiasingTAASharpening</td>
<td>Controls the sharpening filter to help retain sharpness. This is useful when temporal antialiasing introduces blur to a scene during motion.</td>
</tr>
<tr>
<td>Default value: 0.2</td>
<td></td>
</tr>
<tr>
<td>r_AntialiasingTAAUseAntiFlickerFilter</td>
<td>Reduces jitter-based flickering in certain scenarios.</td>
</tr>
<tr>
<td>Valid values: 0 = disables this mode</td>
<td>1 = enables this mode</td>
</tr>
<tr>
<td>r_AntialiasingTAAUseJitterMipBias</td>
<td>Enables mipmap biasing on textures when jitter is enabled. This creates a scene with decreased blur but more flickering.</td>
</tr>
<tr>
<td>Valid values: 0 = disables this mode</td>
<td>1 = enables this mode</td>
</tr>
<tr>
<td>Default value: 1</td>
<td></td>
</tr>
<tr>
<td>r_AntialiasingTAAUseVarianceClamping</td>
<td>Enables variance color clamping to help reduce ghosting. This may increase flickering in the scene. The r_AntialiasingTAAClampingFactor console variable affects only this mode.</td>
</tr>
<tr>
<td>Valid values: 0 = disables this mode</td>
<td>1 = enables this mode</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
</tbody>
</table>

The following images illustrate the range of graphics quality that you can achieve depending on which antialiasing setting you use.
Temporal Antialiasing Limitations

Temporal antialiasing (TAA) is an inexpensive solution that reduces aliasing from all sources and works well with most content. Because it relies on screen-space information from previous frames, TAA has limitations in the following scenarios:

- Objects occlude other objects while in motion. In this scenario, the newly visible areas in the current frame don’t have a history value to blend with and can manifest as ghosting.
- Motion vectors are not present for objects in motion, which can result in subtle smearing artifacts. Motion vectors are not generated for merged vegetation, non-CGF-based particles, or transparent materials. In certain high-frequency signal scenarios, the most apparent artifact is flickering caused by subpixel jittering that alternates between bright and dark pixels.
- The content has a lot of subpixel triangles and normals. In this scenario, the subpixel detail can cause flickering artifacts with temporal subpixel jittering. When a bright edge that is surrounded by dark pixels becomes smaller, the neighborhood clamping heuristic causes the pixels to flicker between light and dark.
- Transparency does not write depth. In this scenario, certain transparent content can exhibit subtle smearing artifacts.

In these cases, you may prefer to use subpixel morphological antialiasing (SMAA) or fast approximate antialiasing (FXAA).

Temporal Antialiasing Best Practices

We recommend following these best practices when using temporal antialiasing (TAA).

- Build content to alias as little as possible.
- Use level of detail (LOD) to reduce subpixel detail for objects that are far away.
- Use the anti-flicker filter to help with flickering. The `r_AntialiasingTAAUseAntiFlickerFilter` console variable is enabled by default.

Controlling Supersampling

In addition to antialiasing, Lumberyard supports supersampling for very high quality rendering. Supersampling renders the scene at a higher resolution and downscales the image to obtain smooth and stable edges. Due to the high internal rendering resolution, supersampling is performance heavy and only suitable for games intended to be played on high-end computers.

Voxel-based Global Illumination (SVOGI)

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Sparse voxel octree global illumination (SVOGI), also known as voxel GI, is a global illumination solution based on voxel ray tracing. It does not require prebaking or manual setup of bounce lights or light volumes. This solution is experimental and may be memory intensive.
Voxel GI provides the following effects:

- Dynamic indirect light bounce from static objects and many dynamic objects.
- Large-scale ambient occlusion (AO) and indirect shadows from static objects such as brushes, terrain, and vegetation.

For every frame, thousands of rays are traced through voxels and shadow maps to gather occlusion and in-directional lighting.

To use this feature, you must enable (p. 1064) the SVOGI gem for your project.

You must also enable SVOGI per level.

**To enable SVOGI for your level**

1. In Lumberyard Editor, choose **Edit, Editor Settings, Global Preferences.**
2. In the Preferences window, under Experimental Features, select Lighting.
3. In the right pane, under Options, select the Total Illumination check box.
4. Click OK.

Integration Modes

You can apply voxel GI through several modes.

Mode 0

With mode 0, only opacity is voxelized. The bounced light is sampled directly from shadow maps—extended to reflective shadow maps—and compute shaders are not used.

Mode 0 has the following advantages:
• GPU memory usage is small (~16 MB).
• Indirect lighting is completely dynamic; moving sun does not cause any slowdown.
• Dynamic objects can bounce indirect lighting.

Mode 0 has the following disadvantages:
• Indirect lighting can have low quality (more noise), especially for small point lights.
• Only single bounce is possible.
• Only diffuse GI is possible.
• Environment probes are needed for specular highlights.

Modes 1, 2

With modes 1 and 2, albedo, normals, and several layers of radiance are voxelized together with opacity. Direct lighting is also injected into voxelization, where it is propagated within the voxelization and then sampled during the ray-tracing pass.

Modes 1 and 2 have the following advantages:
• Modes 1 and 2 support multiple bounces. The light source can be semi-static with multibounce support or be fully dynamic with single bounce support.
• Mode 2 supports traced speculars.
• They provide higher quality, smoother indirect lighting.

Modes 1 and 2 have the following disadvantages:
• Modes 1 and 2 use more GPU memory (64 MB+).
• Large semi-static multibounce lights cannot be moved freely, but moving sun may work fine.
• Dynamic objects cannot affect GI, but can receive it.

Note
If you receive a message that the display driver has stopped responding and has recovered, try this workaround from Microsoft.

Voxel GI Parameters

The following parameters are global for an entire level. You can use normal ambient lights to modulate or tint indirect light intensity locally.

Global illumination uses the sun and the seven largest static lights in the scene and the eight dynamic lights closest to the camera.
To enable voxel global illumination

1. In Lumberyard Editor, choose **Tools, Terrain Tool, Environment**.
2. In the **Environment** panel, under **Total_Illumination_v2**, adjust the following settings as needed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Activate voxel GI for the level.</td>
</tr>
<tr>
<td>Integration mode</td>
<td>Set to 0, 1, or 2:</td>
</tr>
<tr>
<td></td>
<td>• 0 – AO + Sun bounce</td>
</tr>
<tr>
<td></td>
<td>• 1 – Diffuse GI mode (experimental)</td>
</tr>
<tr>
<td></td>
<td>• 2 – Full GI mode (experimental)</td>
</tr>
<tr>
<td></td>
<td>For more information, see <strong>Integration Modes (p. 1820)</strong>.</td>
</tr>
<tr>
<td>Number of bounces</td>
<td>Set the maximum number of indirect bounces. The first indirect bounce is completely dynamic. The remaining bounces are cached in SVO and are mostly static.</td>
</tr>
<tr>
<td>Diffuse cone width</td>
<td>Controls the width of the diffuse cone. Wider cones work faster but may cause overocclusion and increase light leaking.</td>
</tr>
<tr>
<td>Cone max length</td>
<td>Set the maximum length, in meters, of the tracing rays. Shorter rays work faster.</td>
</tr>
<tr>
<td>Use light probes</td>
<td>For integration mode 0, enable this parameter to multiply environment probes lighting with GI. Disable this parameter to replace diffuse contribution of environment probes with GI.</td>
</tr>
<tr>
<td></td>
<td>For integration modes 1 and 2, enable this parameter to use global environment probe for sky light instead of <strong>Time Of Day</strong> fog color.</td>
</tr>
<tr>
<td>Injection multiplier</td>
<td>Modulate light injection by controlling the intensity of bounce light.</td>
</tr>
<tr>
<td>Ambient offset red</td>
<td>Controls the amount of red in the environment. Higher values show more red in the level.</td>
</tr>
<tr>
<td>Ambient offset green</td>
<td>Controls the amount of green in the environment. Higher values show more green in the level.</td>
</tr>
<tr>
<td>Ambient offset blue</td>
<td>Controls the amount of blue in the environment. Higher values show more blue in the level.</td>
</tr>
<tr>
<td>Ambient offset bias</td>
<td>Controls the strength of the light in the environment. Higher values contribute to a brighter environment.</td>
</tr>
<tr>
<td>Saturation</td>
<td>Controls the color saturation of propagated light.</td>
</tr>
<tr>
<td>SSAO amount</td>
<td>Scales down screen space ambient occlusion (SSAO) or screen space directional occlusion (SSDO) amount and radius when GI is active.</td>
</tr>
</tbody>
</table>

3. (Optional) Use the **Voxel Coverage** advanced parameter on each material:
   a. Click **Tools, Material Editor**.
b. Select your material.
c. Under Advanced, modify the **Voxel Coverage** parameter to control the transparency of voxels for each material and manually fix overoccluded areas. The default value is 1.

**Note**
This parameter takes effect only when voxel global illumination is enabled.

### Debugging

You can use the following console variables to help debug voxel GI issues:

- `r_ShowRenderTarget svo_fin` – Displays the output of the voxel GI system.
- `r_profiler 1 | 2` – Retrieves GPU profiling information.
- `svoToggleShowVoxels` – Shows voxelization of the scene, which shows which voxels are on CPU memory and which are on GPU memory.
- `svoReset` – Performs hard reset of the system and recomputes all values.

**Important**
Do not use the `e_svoTI_*` console variables (for example, `e_svoTI_IntegrationMode`) to configure the voxel GI system. Any changes to these console variables in the configuration file will be overwritten by the individual level environment settings file. Instead, configure your settings in the **Total Illumination** (p. 1820) pane in Lumberyard Editor.

### Current Limits

The following limitations exist for the voxel GI system:

- The GI code doesn’t have a mechanism for detecting light modifications directly, but is constantly updating to capture changes to the lighting. This may introduce delay in the GI response to lights changing.
- Large-scale ambient occlusion and indirect shadows are properly cast only by static geometry.
- Voxel GI does not function on some forward-rendering components like particles or water.
- Some artifacts like ghosting, aliasing, light leaking, and noise may be noticeable.
- Procedural vegetation and merged vegetation do not cast occlusion or secondary shadows.
- If a camera is moved to a new location, it may take several seconds until occlusion works properly.
- Only objects and materials with shadow map casting enabled will generate correct bounced light.
- For dynamic objects, indirect light bounce will function only in areas near voxelized static geometry.
- Bounce light may have a noticeable delay of 1 to 2 frames.

Use of the `r_Supersampling=2` console variable may adversely affect the appearance of the voxel GI. You can set the **LowSpecMode** value two times lower to restore the appearance of the voxel GI. **Temporal AA** using `r_AntialiasingMode 2/3` works correctly as well.

### Fog Systems

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Lumberyard supports a standard fog system as well as a voxel-based volumetric fog system. Which one to use for your game comes down to balancing performance over visual quality. Volumetric fog looks superior but comes at a performance cost. The standard fog system is very cheap performance-wise to compute.

You can also add realistic-looking fog above water surfaces, as well as add volumetric fog shadows.

**Topics**
- Standard Fog (p. 1823)
- Volumetric Fog (p. 1825)

## Standard Fog

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard's standard fog system handles sunlight with dynamic shadows and exponential height fog density. However, in dense fog situations the fog's appearance may not be consistent between opaque and transparent materials.

You can add the **Fog Volume** component to an entity to create fog effects. For more information, see the **Fog Volume (p. 593)** component.

**Topics**
- Setting Global (Time of Day) Fog (p. 1823)
- Using Fog Volumes (p. 1823)
- Console Variables for Fog (p. 1824)

## Setting Global (Time of Day) Fog

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use global fog to simulate particles that are distributed uniformly along the ground and fall off exponentially with height above sea level. Global fog accurately accounts for time of day lighting and for scattered sunlight rays to produce haze at the horizon and halos around the sun. You can also use this effect to cast shadows for objects and clouds through the fog.

**To set global fog parameters**

1. In Lumberyard Editor, choose **Tools, Other, Time Of Day**.
2. In the **Parameters** pane, for **Fog**, adjust the **Fog Parameters (p. 1278)** as needed.
3. Close the **Time of Day** editor.

## Using Fog Volumes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Fog volumes are localized 3D areas that define an area where non-volumetric fog is present. Fog volumes do not change in dynamic, nonuniform ways like smoke does. When alpha-transparent objects are behind fog volumes, each pixel is fogged. However, this is not the case when objects are inside fog volumes.

Unlike global (Time of Day) fog that has an upward falloff direction, fog volumes can have an arbitrary falloff direction. Interesting fog shapes and effects can be achieved, including fog patches that vary in size, color, shape, density, and spacing over time, as well as being influenced by wind.

Observe these best practices when creating fog volumes

- Do not overlap fog volumes.
- Make sure indoor fog volumes don't cover more than one sector or they may be culled when the main sector becomes invisible.
- To avoid inaccurate rendering, don't apply nonuniform scaling to fog volumes.
- When using shadow maps inside fog volumes, make sure the environment `VolFogShadows` parameter is disabled.

You can add the **Fog Volume** component to an entity to create fog effects. For more information, see the [Fog Volume](p. 593) component.

**Console Variables for Fog**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/developer-preview) or visit the [AWS Game Tech blog](https://aws.amazon.com/blogs/gaming/) to learn more.

The following console variables can be used to control fog. For more information, see [Using the Console Window](p. 210).

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>e_Fog</code></td>
<td>Toggles fog on and off.</td>
</tr>
<tr>
<td><code>e_FogVolumes</code></td>
<td>Enables local height/distance based fog volumes.</td>
</tr>
<tr>
<td><code>e_FogVolumesTiledInjection</code></td>
<td>Enables tiled fog volume density injection.</td>
</tr>
<tr>
<td><code>r_FogDepthTest</code></td>
<td>Enables per-pixel culling for deferred fog pass. Fog computations for all pixels closer than a given depth value will be skipped.</td>
</tr>
<tr>
<td></td>
<td>0 = culling disabled.</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 = fixed linear world space culling depth</td>
</tr>
<tr>
<td></td>
<td>&lt; 0 = optimal culling depth will be computed automatically based on camera direction and fog settings.</td>
</tr>
<tr>
<td><code>r_FogShadows</code></td>
<td>Enabled deferred volumetric fog shadows.</td>
</tr>
<tr>
<td></td>
<td>0 = No shadows</td>
</tr>
<tr>
<td></td>
<td>1 = Standard resolution</td>
</tr>
</tbody>
</table>
### Fog Systems

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 = Reduced resolution.</td>
</tr>
<tr>
<td>r_FogShadowsMode</td>
<td>Raycasting mode for shadowed fog.</td>
</tr>
<tr>
<td></td>
<td>0 = Brute force shadow map sampling</td>
</tr>
<tr>
<td></td>
<td>1 = Optimized shadow map sampling</td>
</tr>
<tr>
<td>r_FogShadowsWater</td>
<td>Enables volumetric fog shadows over water volumes.</td>
</tr>
</tbody>
</table>

**Volumetric Fog**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Volumetric fog uses volume textures as a view-frustum-shaped voxel buffer to store incoming light and its properties. Volumetric fog supports regular light and sunlight with dynamic shadows, environment probes, ambient light, as well as variations in fog density. It also supports the application of volumetric fog with respect to opaque and transparent materials.

**Example**

In the following example, volumetric fog is disabled.

**Example**

In the following example, volumetric fog is enabled.
You can add the Fog Volume component to an entity to create fog effects. For more information, see the Fog Volume (p. 593) component.

You can specify how light components affect volumetric fog. For more information, see Light Component Fog Properties (p. 597).

You can also use the Particle Editor to place a particle emitter in your level to add fog density to an area. For more information, see the Volume Fog and Volume Thickness parameters for the Advanced Attribute in the Particles Attributes Reference (p. 1513).

To add localized nonvolumetric regions of fog, see Using Fog Volumes (p. 1823).

Topics
- Best Practices for Volumetric Fog (p. 1826)
- Setting Global (Time of Day) Volumetric Fog (p. 1827)
- Console Variables for Volumetric Fog (p. 1827)

Best Practices for Volumetric Fog

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

See the following best practices for working with volumetric fog.

- Attach a Fog Volume (p. 593) and a Box Shape (p. 818) component to the entity.
- Set the r_DeferredShadingTiled console variable to a value greater than 0. The recommended value is 1 to 2. This is required to use volumetric fog. For more information, see Using the Console Window (p. 210).
- To avoid performance problems, use the default values for the Ramp Start and Ramp End parameters located in the Time of Day editor. For more information, see Setting Global (Time of Day) Volumetric Fog (p. 1827).
In the **Time of Day Editor**, large values for the **Range** parameter can cause fog flicker and light leaking behind walls.

To correct this effect, you can adjust the value for the `r_VolumetricFogTexDepth` console variable accordingly.

**Example**

The default value for the `r_VolumetricFogTexDepth` console variable is 32 and the default value for **Range** is 64. If you want to use a larger range such as **Range**=256 and maintain the same visual quality, you need to set `r_VolumetricFogTexDepth` to 64.

When **Range** is 1024, set `r_VolumetricFogTexDepth` to 128.

### Setting Global (Time of Day) Volumetric Fog

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use global volumetric fog to simulate particles that are distributed uniformly along the ground and fall off exponentially with height above sea level. Global volumetric fog accurately accounts for time of day lighting and for scattered sunlight rays to produce halos around the sun.

Use the **Anisotropy** parameters to control the amount of sunlight that is scattered through fog and the direction. Set the **Anisotropy (atmosphere)** parameter close to 0 to achieve a uniform look across the entire sky. Set the **Anisotropy (sun radial)** parameter close to 1 to create a bloom effect around the sun.

The **Radial blend** parameters blend the **Anisotropy** parameters to create various effects. For example, you can create sun radial scattering by setting **Radial blend mode** to 1 and **Radial blend factor** to 1.

**To set global volumetric fog parameters**

1. In Lumberyard Editor, click **Tools, Other, Time Of Day**.
2. In the **Parameters** pane, for **Volumetric fog**, adjust the **Volumetric Fog Parameters** as needed.
3. Close the **Time of Day** editor.

### Console Variables for Volumetric Fog

You can specify the following console variables (CVARs) to enable volumetric fog effects. For more information, see Using the Console Window (p. 210).

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>e_VolumetricFog</code></td>
<td>Toggles the <strong>Volumetric Fog</strong> component on or off</td>
<td>0 = Off,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On</td>
</tr>
<tr>
<td>Console Variable</td>
<td>Description</td>
<td>Values</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>r_VolumetricFogTexScale</td>
<td>Sets the width and height of the internal volume texture. Screen resolution divided by this factor is applied to the width and height of the fog volume.</td>
<td>The minimum value should be 2.</td>
</tr>
<tr>
<td>r_VolumetricFogTexDepth</td>
<td>Sets the depth for the internal volume texture.</td>
<td>This value should be multiples of 4, but less than 252.</td>
</tr>
<tr>
<td>r_VolumetricFogReprojectionBlendFactor</td>
<td>Sets the blending factor of the temporal reprojection filter. Higher values cause less flicker, but more ghosting.</td>
<td>0 = Turn off temporal reprojection.</td>
</tr>
<tr>
<td>r_VolumetricFogReprojectionMode</td>
<td>Sets the mode of ghost reduction for the temporal re-projection filter.</td>
<td>0 = Conservative mode. Ghost artifacts can appear when a light moves, but there is less flicker than in the advanced mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Advanced mode. This mode has fewer ghost artifacts but slightly more flicker than the conservative mode.</td>
</tr>
<tr>
<td>r_VolumetricFogSample</td>
<td>Sets the number of sample points.</td>
<td>0 = 1 sample point in a voxel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = 2 sample points in a voxel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = 4 sample points in a voxel.</td>
</tr>
<tr>
<td>r_VolumetricFogShadow</td>
<td>Sets the shadow sample count for each sample point.</td>
<td>0 = 1 shadow sample per sample point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = 2 shadow samples per sample point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = 3 shadow samples per sample point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = 4 shadow samples per sample point.</td>
</tr>
</tbody>
</table>
### Console Variable

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_VolumetricFogDownscaledSunShadow</td>
<td>Replaces sun shadow maps with downscaled shadow maps or static shadow maps, if possible. This reduces the volumetric fog flicker for sun shadow.</td>
<td>0 = Disabled.</td>
</tr>
<tr>
<td></td>
<td>1 = Replace first and second cascades with downscaled shadow maps. Others are replaced with a static shadow map, if possible.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 = Replace first, second, and third cascades with downscaled shadow maps. Others are replaced with a static shadow map, if possible.</td>
<td></td>
</tr>
<tr>
<td>r_VolumetricFogDownscaledSunShadowRatio</td>
<td>Sets the downscale ratio for sun shadow maps.</td>
<td>0 = 1/4 downscaled sun shadow maps.</td>
</tr>
<tr>
<td></td>
<td>1 = 1/8 downscaled sun shadow maps.</td>
<td>1 = 1/8 downscaled sun shadow maps.</td>
</tr>
<tr>
<td></td>
<td>2 = 1/16 downscaled sun shadow maps.</td>
<td>2 = 1/16 downscaled sun shadow maps.</td>
</tr>
<tr>
<td>r_VolumetricFogMinimumLightBulbSize</td>
<td>Sets the minimum size threshold for light attenuation bulb size for voxel-based volumetric fog.</td>
<td>Specify a value between 0 to 2.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Small bulb sizes can cause light flicker.</td>
<td>Default value: 0.4</td>
</tr>
</tbody>
</table>

### Render Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To visualize objects in a world, Lumberyard defines the concepts of the render node and render element. Render nodes represent general objects in the 3D engine. Among other things, they are used to build a hierarchy for visibility culling, allow physics interactions (optional), and rendering.
For actual rendering, render nodes add themselves to the renderer, passing an appropriate render
element that implements the actual drawing of the object. This process happens with the help of render
objects, as shown in the sample code below

Creating a New Render Node

The following example creates a render node called **PrismObject**. It is derived from **IRenderNode**,
defined in Code/CryEngine/CryCommon/IEntityRenderState.h.

1. Add the interface for **IPrismObjectRenderNode** to CryEngine/CryCommon/
   IEntityRenderState.h to make it publicly available.

   ```cpp
   struct IPrismRenderNode : public IRenderNode
   {
       ...
   };
   ```

2. Add a new enum to the list of already defined render nodes in CryEngine/CryCommon/
   IEntityRenderState.h.

   ```cpp
   enum EERType
   {
       ...
       eERType_PrismObject,
       ...
   };
   ```

3. Add **PrismObjectRenderNode.h** to Cry3DEngine.

   ```cpp
   #ifndef _PRISM_RENDERNODE_
   #define _PRISM_RENDERNODE_
   #pragma once

   class CPrismRenderNode : public IPrismRenderNode, public Cry3DEngineBase
   {
   public:
       // interface IPrismRenderNode
       ...

       // interface IRenderNode
   virtual void SetMatrix(const Matrix34& mat);
   virtual EERType GetRenderNodeType();
   virtual const char* GetEntityClassName() const { return "PrismObject"; }
   virtual const char* GetName() const;
   virtual Vec3 GetPos(bool bWorldOnly = true) const;
   virtual bool Render(const SRendParams &rParam);
   virtual IPhysicalEntity* GetPhysics() const { return 0; }
   virtual void SetPhysics(IPhysicalEntity*) {}
   virtual void SetMaterial(IMaterial* pMat) { m_pMaterial = pMat; }
   virtual IMaterial* GetMaterial(Vec3* pHitPos = 0) { return m_pMaterial; }
   virtual float GetMaxViewDist();
   virtual void GetMemoryUsage(ICrySizer* pSizer);
   virtual const AABB GetBBox() const { return m_WSBBox; }
   virtual void SetBBox( const AABB& WSBBox ) { m_WSBBox = WSBBox; }
   
   private:
   CPrismRenderNode();

   private:
   ~CPrismRenderNode();
   
   #endif  
   ```
4. Add PrismObjectRenderNode.cpp to Cry3DEngine.

```cpp
#include "StdAfx.h"
#include "PrismRenderNode.h"

CPrismRenderNode::CPrismRenderNode() : m_pMaterial(0)
{
  m_mat.SetIdentity();
  m_WSBBox = AABB(Vec3(-1, -1, -1), Vec3(1, 1, 1));
  m_pRE = (CREPrismObject*) GetRenderer()->EF_CreateRE(eDATA_PrismObject);
  m_dwRndFlags |= ERF_CASTSHADOWMAPS | ERF_HAS_CASTSHADOWMAPS;
}

CPrismRenderNode::~CPrismRenderNode()
{
  if (m_pRE)
    m_pRE->Release(false);
  Get3DEngine()->FreeRenderNodeState(this);
}

void CPrismRenderNode::SetMatrix(const Matrix34& mat)
{
  m_mat = mat;
  m_WSBBox.SetTransformedAABB(mat, AABB(Vec3(-1, -1, -1), Vec3(1, 1, 1)));
  Get3DEngine()->RegisterEntity(this);
}

const char* CPrismRenderNode::GetName() const
{
  return "PrismObject";
}

void CPrismRenderNode::Render(const SRendParams& rParam, const SRenderingPassInfo &passInfo)
{
  FUNCTION_PROFILER_3DENGINE;
  if(!m_pMaterial)
    return;

  // create temp render node to submit this prism object to the renderer
  CRenderObject *pRO = GetRenderer()->EF_GetObject_Temp(passInfo.ThreadID());                     // pointer could be cached
  if(pRO)
  {
    // set basic render object properties
    pRO->m_II.m_Matrix = m_mat;
    pRO->m_ObjFlags |= FOB_TRANS_MASK;
    pRO->m_fSort = 0;
    pRO->m_fDistance = rParam.fDistance;

    // transform camera into object space
    const CCamera& cam(passInfo.GetCamera());
    Vec3 viewerPosWS(cam.GetPosition());

```
// set render object properties
m_pRE->m_center = m_mat.GetTranslation();
SShaderItem& shaderItem(m_pMaterial->GetShaderItem(0));
GetRenderer()->EF_AddEf(m_pRE, shaderItem, pRO, passInfo,
EFSLIST_GENERAL, 0, SRendItemSorter(rParam.rendItemSorter));
}

void CPrismRenderNode::GetMemoryUsage(ICrySizer* pSizer) const
{
    SIZER_COMPONENT_NAME(pSizer, "PrismRenderNode");
pSizer->AddObject(this, sizeof(*this));
}

void CPrismRenderNode::OffsetPosition(const Vec3& delta)
{
    if (m_pRNTmpData) m_pRNTmpData->OffsetPosition(delta);
    m_WSBBox.Move(delta);
    m_mat.SetTranslation(m_mat.GetTranslation() + delta);
    if (m_pRE) m_pRE->m_center += delta;
}

void CPrismRenderNode::FillBBox(AABB & aabb)
{
    aabb = CPrismRenderNode::GetBBox();
}

EERType CPrismRenderNode::GetRenderNodeType()
{
    return eE>Type_PrismObject;
}

float CPrismRenderNode::GetMaxViewDist()
{
    return 1000.0f;
}

Vec3 CPrismRenderNode::GetPos(bool bWorldOnly) const
{
    return m_mat.GetTranslation();
}

IMaterial* CPrismRenderNode::GetMaterial(Vec3* pHitPos)
{
    return m_pMaterial;
}

5. To allow client code to create an instance of the new render node, extend the following function in /
   Code/CryEngine/Cry3DEngine/3DEngine.cpp

... #include "PrismRenderNode.h"
...
IRenderNode * C3DEngine::CreateRenderNode(EERType type)
{
    switch (type)
    {
    ...
    case eE>Type_PrismObject:
    {
        IPrismRenderNode* pRenderNode = new CPrismRenderNode();
        return pRenderNode;
    }
Generating Stars .dat File

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Stars .dat file contains star data that is used in sky rendering. This topic provides information you'll need if you want to modify the data in this file. It assumes you have some familiarity with generating binary files.

Star data is located in `Build\Engine\EngineAssets\Sky\stars.dat`. This data is loaded in the function `CStars::LoadData`, implemented in the file `CRESky.cpp`.

### File Format

The Stars .dat file uses a simple binary format; it can be easily modified using an editing tool. The file starts with a header, followed by entries for each star. The header specifies the number of entries in the file.

All types stored in little-endian format, float32 in IEEE-754 format.

Star data provided in the SDK is based on real-world information. Typically, you can also use existing star catalogs to populate this information for you.

The file elements are as follows:

#### Header (12 bytes)

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>0</td>
<td>uint32</td>
<td>0x52415453 (ASCII: STAR)</td>
</tr>
<tr>
<td>Version</td>
<td>4</td>
<td>uint32</td>
<td>0x00010001</td>
</tr>
<tr>
<td>NumStars</td>
<td>8</td>
<td>uint32</td>
<td>Number of star entries in the file</td>
</tr>
</tbody>
</table>

#### Entry (12 bytes)

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RightAscension</td>
<td>0</td>
<td>float32</td>
<td>in radians</td>
</tr>
<tr>
<td>Declination</td>
<td>4</td>
<td>float32</td>
<td>in radians</td>
</tr>
<tr>
<td>Red</td>
<td>8</td>
<td>uint8</td>
<td>star color, red channel</td>
</tr>
<tr>
<td>Green</td>
<td>9</td>
<td>uint8</td>
<td>star color, green channel</td>
</tr>
<tr>
<td>Blue</td>
<td>10</td>
<td>uint8</td>
<td>star color, blue channel</td>
</tr>
<tr>
<td>Magnitude</td>
<td>11</td>
<td>uint8</td>
<td>brightness, normalized range</td>
</tr>
</tbody>
</table>
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes a preview of DirectX 12, which provides the framework for you to build DirectX 12 supported applications and develop features. DirectX 12 provides the following benefits for developing applications:

- Slightly faster overall GPU performance on NVIDIA cards.
- Slight increase in frame rate for scenes that are GPU-bound.
- Supported features from DirectX 12 and DirectX 11.3 that include the following:
  - Rasterizer-ordered view with order independent transparency.
  - Tiled resources that allow you to develop textures virtually that you can then use on terrain.
  - Optimized texture streaming.
  - Conservative rasterization helps you develop and optimize ray tracing or voxel rendering, which you can use to develop certain rendering techniques such as global illumination (GI).

Based on your scene, you may discover faster or slower performance across DirectX 11 and DirectX 12. For example, if you enable DirectX 12 you may see a slight performance degradation with CPU-bound scenes. Scenes with high draw calls may also see slower performance depending on the size of the scene and view distance.

**To enable DirectX 12 rendering in Lumberyard**

1. On Windows 10, install the latest Windows 10 SDK.
   
   **Note**
   Copy the Windows 10 SDK version. In Windows, you can find this value in Programs and Features.

2. In a text editor, open the `system_windows_pc.cfg` file and add the following parameter:

   ```
   r_Driver = DX12
   ```

3. Configure your Lumberyard project by doing one of the following:

   - With a text editor, open the `user_settings.options` file in the `lumberyard_version\dev\waf` directory and edit the following parameters, removing any leading semicolons to uncomment the line:
     - Under the [Build Options] section, set `win_build_renderer` to `DX12`.
     - Under the [Windows Options] section, set `win_vs2017_winkit` to your version of the Windows 10 SDK (for example, `10.1.17134.12`).

   Navigate to the `lumberyard_version\dev` directory and, in a command line window, enter the following:

   ```
   lmbr_waf configure
   ```

   - Or, with a command line window, navigate to the `lumberyard_version\dev` directory and enter the following:

   ```
   lmbr_waf configure --win-build-renderer=DX12 --win-vs2017-winkit=10.1.17134.12
   ```
4. In a command line window, build your project with the following command:

```
lmbr_waf build_win_x64_vs2017_profile -p game_and_engine
```

5. Start your game with the launcher. You can verify that your game has enabled DX12 in the top-right of the viewport.

**Note**

- If the DirectX version doesn't appear in the viewport, verify that the `r_DisplayInfo` console variable is set to 1 or 2. For more information, see Using the Console Window (p. 210).
- Currently, Lumberyard Editor does not support DirectX 12. If you open the game in the editor, the editor defaults to DirectX 11.

For more information about DirectX 12, see Taking Advantage of DirectX 11.2 Tiled Resources and Don't Be Conservative with Conservative Rasterization.
Programming in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Programming with Lumberyard

Learn about the game development APIs and integrations available with Lumberyard (and for Lumberyard). This documentation covers the EBus messaging infrastructure for in-game communications between game systems, memory management and debugging, extending and customizing the Lumberyard editor and tools, networking, localization, and Twitch integration.

These topics are for game developers who have experience with C++ programming and common game design patterns.

In this section:

<table>
<thead>
<tr>
<th>Topic Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems APIs (p. 1837)</td>
<td>Learn about the AZ C++ APIs used for console variables, event notifications, and simple pub/sub messaging within your game project. Note: These systems APIs were introduced with Lumberyard 1.24 and focus on a simplified model for in-game messaging and events.</td>
</tr>
<tr>
<td>Event Bus (p. 1851) (EBus)</td>
<td>Learn about EBus, the messaging infrastructure for notifications and messaging.</td>
</tr>
<tr>
<td>AZ Code Generator (p. 1869)</td>
<td>Learn about AZ Code Generator, a utility for generating source code (or any data or text) from specially tagged source code.</td>
</tr>
<tr>
<td>Input (p. 1902)</td>
<td>Documentation on incorporating different control devices and configurations for your Lumberyard game.</td>
</tr>
<tr>
<td>Memory management (p. 1906)</td>
<td>Learn how Lumberyard allocates and manages memory resources for games, as well as memory debugging and overrun detection.</td>
</tr>
<tr>
<td>Profiling, testing, and debugging (p. 1920)</td>
<td>Learn about Lumberyard tools that are used for testing builds, profiling performance, and debugging various issues that may be encountered.</td>
</tr>
<tr>
<td>UI 2.0 (p. 1973)</td>
<td>Learn about extending the Lumberyard editor and tools user interface with UI 2.0 and Qt. Note: This documentation was introduced with Lumberyard 1.25, and will be updated with the full set of guidance for the next release.</td>
</tr>
<tr>
<td>Networking (p. 1990)</td>
<td>Learn about GridMate, Lumberyard's game networking infrastructure. Learn about how to synchronize game state across clients, manage bandwidth usage, provide data encryption over the wire, and integrate with Amazon GameLift (for multiplayer lobbies and matchmaking).</td>
</tr>
<tr>
<td>Twitch integration (p. 2095)</td>
<td>Learn how to incorporate Twitch into your Lumberyard game project.</td>
</tr>
<tr>
<td>Topic Area</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cloud connected features</td>
<td>Learn about Cloud Canvas, a set of gems, scripting features, and resource groups that you can use with AWS Cloud Services and Lumberyard to create cloud-enabled games.</td>
</tr>
<tr>
<td>Localization</td>
<td>Learn how to create multiple-language, region-specific versions of your game using Lumberyard’s localization system.</td>
</tr>
</tbody>
</table>

**Working with Amazon Lumberyard systems APIs**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Amazon Lumberyard provides a number of APIs to access and invoke key parts of the Lumberyard engine from your game, including a new in-game console and a new event management system.

These C++ APIs are called from your code, and provide important infrastructure functionality for your game, including messaging and communications, and an in-game console with variable and functor support.

You can find these APIs defined in the following headers:

- Event.h (in %INSTALL_ROOT%\dev\Code\Framework\AzCore\AzCore\EBus\) for AZ::Event (p. 1842)
- IConsole.h (in %INSTALL_ROOT%\dev\Code\Framework\AzCore\AzCore\IConsole\) for AZ::Console (p. 1837)
- Interface.h (in %INSTALL_ROOT%\dev\Code\Framework\AzCore\AzCore\Interface\) for AZ::Interface (p. 1846)

Some of these systems APIs replace functionality from older versions, or provide alternative methods, such as AZ::Console for the CryConsole and AZ::Event for EBus.

For C++ API reference documentation, see the Amazon Lumberyard C++ API Reference.

**Topics**

- AZ::Console (p. 1837)
- AZ::Event<...> (p. 1842)
- AZ::Interface<T> (p. 1846)

**AZ::Console**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The AZ::Console class provides a set of macros for defining variables and mapping functions that you can use to interact with in-game variables and processes. Use the macros defined in this class to set the console variables (cvars) and functors (cfuncs) for your game, and then access them through the Lumberyard console.
AZ::Console is defined in the following header: %INSTALL-ROOT%\dev\Code\Framework\AzCore\AzCore\Console\IConsole.h

Note
The console found inside AzCore is a console and cvar system intended to replace the legacy CryEngine console and cvar system. This system is free of any CryEngine code, and depends only on AzCore.

AZ::Console features:

- **Stubbed** support for multiplayer (as of v1.24). Lumberyard will eventually enable cvar replication across multiplayer instances using GridMate.
- Basic access protections and anti-cheat mechanisms for locking down cvs and cfuncs in release builds.
- Default support for several C++ types, including bool (Boolean), stdint (all types), floats, doubles, vectors and quaternions, and enums (enumerations).
- Flexible and expandable type support. You can add support for new cvar types without altering the console code directly.
- Support for the legacy CryEngine console, specifically to reduce user confusion. For users of the in-game console, legacy cvs and AZ_CVAR cvs appear identical.

Important
As of Amazon Lumberyard version 1.24, AZ::Console does not support the following features present in the CryEngine console:
• Registration of cvars at runtime by name. Formerly, you could register a CryEngine cvar using `IConsole->Register<String/Int/Float>()` and then retrieve the value of that cvar with `IConsole->GetCVar()`. AZ::Console cvars must be declared at compile-time using one of the macros in `IConsole.h`.

• Custom cvar/cfunc autocomplete callbacks. Formerly, you could specify an autocomplete callback and attach it to a CryEngine cvar. AZ::Console currently does not support this functionality.

• Remote console support. The CryEngine console allows remote access by opening a socket on a provided port, and then listening for console commands issued to that socket. AZ::Console currently does not support this functionality.

Topics

- Console variables (cvars) (p. 1839)
- Console functors (cfuncs) (p. 1840)
- Optional flags (p. 1841)
- Adding support for new console variable types (p. 1842)

Console variables (cvars)

Declare a cvar using one of two macros from `IConsole.h`:

```
AZ_CVAR(_TYPE, _NAME, _INIT, _CALLBACK, _FLAGS, _DESC) //Standard cvar macro, provides no external linkage.
```

```
AZ_CVAR_EXTERNABLE(_TYPE, _NAME, _INIT, _CALLBACK, _FLAGS, _DESC) //Cvar macro that creates a console variable with external linkage.
```

Parameters:

- **_TYPE**: The base type of the cvar.
- **_NAME**: The name of the cvar.
- **_INIT**: The initial value to assign to the cvar.
- **_CALLBACK**: An optional callback function invoked when a cvar changes value. **Note**: These macros do not guarantee that this callback will be run on a specific thread. The implementor of the callback handler is responsible for ensuring thread safety.
- **_FLAGS**: One or more AZ::Console::FunctorFlags that are used to mutate behavior. Use the logical AND (`&&`) and OR (`||`) operators to combine flags. If you do not have any flags to set, use `FunctorFlags::None`.
- **_DESC**: String that provides a short description of the cvar for display.

To declare a new cvar in your code, include the `IConsole.h` header. Then use one of the cvar macros (such as `AZ_CVAR`) to declare your new console variable in your own code (.cpp) files.

**Note**

AZ_CVAR and AZ_CVAR_EXTERNABLE variables can be declared only in C++ code (.cpp) files. AZ_CVAR_EXTERNED variables, however, can be declared in either C++ code (.cpp) or header (.h) files.

Here are some examples.
AZ_CVAR(int32_t, cl_GameServiceRefreshTimeMs, 1000, nullptr, FunctorFlags::None, "Controls the auto-refresh delay for all gameService data, time in milliseconds");
AZ_CVAR(bool, cl_QuitOnHubDisconnect, false, nullptr, FunctorFlags::None, "If enabled, the client executable will terminate on disconnect");

void OnConsoleResUpdate(const int32_t& a_NewWidth)
{
    // Run update for new value
}
AZ_CVAR(int32_t, sv_ConsoleWidth, 160, OnConsoleResUpdate, FunctorFlags::ReadOnly, "The width of the server console window");

AZ_CVAR_EXTERNABLE(uint16_t, net_ServerRateMs, 33, nullptr, FunctorFlags::ReadOnly, "Server tick rate to use for network relevant simulations");

Optionally, use the following name prefixes to help organize groups of cvars:
- sv_: For server only cvars
- cl_: For client only cvars
- bg_: "Both games" for common cvars (client and server)

These prefixes are useful to quickly limit the scope of autocomplete, and to see groups of associated cvars in the console. You can use your own prefixes as well.

To make an existing console variable external (extern), use the AZ_CVAR_EXTERNED macro:

```
AZ_CVAR_EXTERNED(_TYPE, _NAME)
```

Make sure that the _TYPE and _NAME parameters match those of the previously defined cvar.

**Console functors (cfuncs)**

Console functions allow you to register a command with the console that's not associated with a specific type or value. In Lumberyard, they're purely a mechanism to allow a method to be invoked directly from the Lumberyard in-game console.

There are two types of cfuncs: one to invoke class member methods (AZ_CONSOLESFUNC), and one to invoke static methods (AZ_CONSOLEFREEFUNC).

To declare a class member method cfunc, use the AZ_FUNC macro from IConsole.h:

```
AZ_CONSOLESFUNC(_CLASS, _FUNCTION, _INSTANCE, _FLAGS, _DESC)
```

Parameters:
- _CLASS: The class that contains the method (function) for invocation.
- _FUNCTION: The method to invoke as a callback. **Note:** These macros do not guarantee that this callback will be run on a specific thread. The implementor of the callback handler is responsible for ensuring thread safety.
• _INSTANCE: The instance of the class on which this method gets invoked (usually set to this for the current instance).
• _FLAGS: One or more AZ::Console::FunctorFlags that are used to mutate behavior. Use the logical AND (&) and OR (|) operators to combine flags. If you do not have any flags to set, use FunctorFlags::None.
• _DESC: String that provides a short description of the cfunc for display.

Some examples of cfunc declarations:

```cpp
class Example
{
    public:
        Example() { AZ_CONSOLEFUNC(Example, Method, this, FunctorFlags::DontReplicate, 
            "Executes the Method method on this Example instance, invoke in the console using 
            Example.Method"); }
        void Method(const StringSet&) {}
};
```

To declare a cfunc for a static method (or other non-member function), use the AZ_CONSOLEFREEFUNC macro:

```cpp
AZ_CONSOLEFREEFUNC(_FUNCTION, _FLAGS, _DESC)
```

Parameters:
• _FUNCTION: The static method to invoke as a callback. Note: These macros do not guarantee that this callback will be run on a specific thread. The implementor of the callback handler is responsible for ensuring thread safety.
• _FLAGS: One or more AZ::Console::FunctorFlags that are used to mutate behavior. Use the logical AND (&) and OR (|) operators to combine flags. If you do not have any flags to set, use FunctorFlags::None.
• _DESC: String that provides a short description of the cfunc for display.

Example:

```cpp
void ForceEnableMetrics(const StringSet&) {}
AZ_CONSOLEFREEFREEFUNC(ForceEnableMetrics, FunctorFlags::Null, "If called, force enable metrics");
```

Optional flags

AZ::Console provides a set of flags that can be passed to cvar and cfunc declarations and indicate how they should be handled:

```cpp
enum class FunctorFlags
{
    Null = 0, // Empty flags
    DontReplicate = (1 << 0), // Should not be replicated (CURRENTLY UNUSED)
    ServerOnly = (1 << 1), // Should never replicate to clients (CURRENTLY UNUSED)
    ReadOnly = (1 << 2), // Should not be invoked at runtime
    IsCheat = (1 << 4), // Should not be shown in the console for autocomplete
    IsDeprecated = (1 << 5), // Command is deprecated, show a warning when invoked
    NeedsReload = (1 << 6), // Level should be reloaded after executing this command
};
```
Adding support for new console variable types

To add support for a new cvar type, override the two template methods that convert the custom type to a space-delimited string from a vector of space-delimited string inputs.

As an example, an override that converts AZ::Vector3 to a string and back to a value is declared like this:

```cpp
namespace AZ
{
    // CVar compatibility
    namespace ConsoleTypeHelpers
    {
        template <>
        AZStd::string ValueToString<AZ::Vector3>(const AZ::Vector3& a_Value);
        template <>
        bool StringSetToValue<AZ::Vector3>(AZ::Vector3& a_OutValue, const StringSet& a_Arguments);
    }
}
```

AZ::Event<...>

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The AZ::Event template class is used to subscribe to and publish single value messages across the different components of your game. It's designed to replace value-based event pub/sub patterns that are currently implemented using EBus, only with significantly simpler syntax. There are a number of benefits to this new system, including simpler code, fewer files, removal of aggregate interfaces where a handler only cares about a subset of events, and improved runtime performance when dispatching value changes to registered handlers.

AZ::Event is defined as a C++ template (template <typename... Params>) in the following header: %INSTALL-ROOT%\dev\Code\Framework\AzCore\AzCore\Ebus\Event.h

Important
As of Amazon Lumberyard version 1.24, AZ::Event limitations include the following:

- The event system is single-threaded only. Handlers should Connect() and Disconnect() on the same thread that is dispatching events.
- Handlers can be bound only to an existing event instance. You can't bind to an event prior to its creation (the way you can with an address by ID EBus).
- A handler can be bound only to a single event. You can't bind a single handler to more than one event.
- There are no return results for handlers. The handler function signature must have a void return result.
- There is no event queuing. A queue can be built as a modular handler wrapper, but in the single-threaded implementation, all events immediately dispatch to all handlers.
AZ::Event provides a Handler class and the following explicit constructors:

- Handler(std::nullptr_t)
- Handler(Callback callback)
- Handler(const Handler& rhs)
- Handler(Handler&& rhs)

AZ::Event::Handler has the following methods defined on it:

- To connect to a Handler instance: void Connect(Event<Params...>& event);
- To disconnect from a Handler instance: void Disconnect();

Example usage

- To create an event for handling, declare an instance of AZ::Event with the following C++ syntax:
  
  AZ::Event<{type}> {name_of_event};

  For example, to declare an event that can publish a Boolean value:

  AZ::Event<bool> isPlayerActive;

- To declare a handler that will process the event when it is signaled:

  AZ::Event<bool>::Handler playerActiveHandler([]({type} value) {});

  For example, to create a handler for the event from the previous example:

  AZ::Event<bool>::Handler playerActiveHandler([](bool value) {});

When you declare the event and the handler in your header, you can connect to the event and signal it. Here is a simple example using the declarations and calls from the prior examples:

```cpp
// Declaration in your header
AZ::Event<bool> isPlayerActive; // Declare the event
AZ::Event<bool>::Handler playerActiveHandler([](bool value) {}); // Declare our handler

// Usage in your code
handler.Connect(isPlayerActive); // Connect the handler to our event
// ...
isPlayerActive.Signal(true); // Signal the event to inform subscribers that the player is active
```

Here is a more complex example that signals multiple events with a class to handle them:

```cpp
class ExampleEventComponent
    : public AZ::Component
{
    public:
        using Event1Type = AZ::Event<const AZ::Vector3&>;
        using Event2Type = AZ::Event<float, float>

        void Tick()
        {
            // Update component state
            if (value1Changed)
            {
                m_event1.Signal(value1);
            }
```
if (value2Changed)
    {
        m_event2.Signal(value2.x, value2.y);
    }
}

void ConnectEvent1Handler(Event1Type::Handler& handler) { handler.Connect(m_event1); }
void ConnectEvent2Handler(Event2Type::Handler& handler) { handler.Connect(m_event2); }
private:
    Event1Type m_event1;
    Event2Type m_event2;
};
class ExampleHandlerComponent
 : public AZ::Component
{
public:
    ExampleHandlerComponent()
        : m_handler1([this](const AZ::Vector3& value) { this->OnEvent1Invoked(value); }),
          m_handler2([this](float value2x, float value2y) { this->m_value2x = value2x; this->m_value2y = value2y; })
    {
    }

    void Activate()
    {
        ExampleEventComponent* eventComponent = GetEntity()->FindComponent<ExampleEventComponent>();
        if (eventComponent)
        {
            eventComponent->ConnectEvent1Handler(m_handler1);
            eventComponent->ConnectEvent2Handler(m_handler2);
        }
    }

    void OnEvent1Invoked(int32_t value) { // do something with value }
private:
    ExampleEventComponent::Event1Type::Handler m_handler1;
    ExampleEventComponent::Event2Type::Handler m_handler2;
};

Performance

AZ::Event is roughly another 20% faster than even the lambda syntax for EBus, and over 40% faster than EBus's member function pointer model. These performance deltas scale linearly with the number of handlers, so AZ::Event is 40% faster than using standard EBus member function pointers whether there's 1,000 handlers attached, or 1,000,000.

To compare the EBus handler implementation code against AZ::Event, here is an example of code used to signal a change to a single value using EBus.

// Single-value message handler using EBus

// Bus interface
class EBusEventExample
 : public AZ::EBusTraits
{
public:
    using MutexType = NullMutex;
    static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Multiple;
    static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
    virtual void OnSignal(int32_t) = 0;
using EBusEventExampleBus = AZ::EBus<EBusEventExample>;

class EBusEventExampleImpl : public EBusPerfBaselineBus::Handler
{
public:

    EBusEventExampleImpl() { EBusEventExampleBus::Handler::BusConnect(); }
    ~EBusEventExampleImpl() { EBusEventExampleBus::Handler::BusDisconnect(); }

    void OnSignal(int32_t) override {};
};

// Usage
EBusEventExampleImpl handler;
EBusEventExampleBus::Broadcast(&EBusEventExample::OnSignal, 1);

And here is an example that performs the same work using AZ::Event.

// Single-value message handler implemented using AZ::Event
AZ::Event<int32_t> event; // Declare the event
AZ::Event<int32_t>::Handler handler([](int32_t value) {}); // Declare our handler

// Usage
handler.Connect(event); // Connect the handler to our event
event.Signal(1); // Signal an event, this will invoke our handler's lambda

Note the reduced lines of code, as well as the overall simpler code pattern. Try it out by porting some of your current EBus message handlers to use AZ::Event, and then test it using our built-in unit tests and benchmarks.

Unit testing and benchmarking

The AZ::Event system includes a number of unit tests and benchmarks to validate correct behavior and confirm the performance advantages over an equivalent EBus implementation.

To execute the unit tests, the following command-line arguments can be provided to the AzTestRunner:

%INSTALL-ROOT%\dev\Bin64\vc141.Test\AzCoreTests.dll AzRunBenchmarks --pause-on-completion --benchmark_filter=BM_EventPerf*

You should see unit testing output like this.

[==========] Running 7 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 7 tests from EventTests
[ RUN      ] EventTests.TestHasCallback
[   OK ] EventTests.TestHasCallback (0 ms)
[ RUN      ] EventTests.TestScopedConnect
[   OK ] EventTests.TestScopedConnect (0 ms)
[ RUN      ] EventTests.TestEvent
[   OK ] EventTests.TestEvent (1 ms)
[ RUN      ] EventTests.TestEventMultiParam
[   OK ] EventTests.TestEventMultiParam (0 ms)
[ RUN      ] EventTests.TestConnectDuringEvent
[   OK ] EventTests.TestConnectDuringEvent (0 ms)
[ RUN      ] EventTests.TestDisconnectDuringEvent
[   OK ] EventTests.TestDisconnectDuringEvent (0 ms)
[ RUN      ] EventTests.TestDisconnectDuringEventReversed
[   OK ] EventTests.TestDisconnectDuringEventReversed (1 ms)
To execute the benchmarks, the following command-line arguments can be provided to the AzTestRunner:

```bash
%INSTALL-ROOT%\dev\Bin64vc141.Test\AzCoreTests.dll AzRunBenchmarks --pause-on-completion --benchmark_filter=BM_EventPerf*
```

You should see benchmark output like this.

<table>
<thead>
<tr>
<th>Benchmark name</th>
<th>benchmark time</th>
<th>cpu time</th>
<th>iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_EventPerf_EventEmpty</td>
<td>16869 ns</td>
<td>16881 ns</td>
<td>40727</td>
</tr>
<tr>
<td>BM_EventPerf_EventIncrement</td>
<td>20124 ns</td>
<td>20508 ns</td>
<td>37333</td>
</tr>
<tr>
<td>BM_EventPerf_EBusEmpty</td>
<td>29421 ns</td>
<td>29157 ns</td>
<td>23579</td>
</tr>
<tr>
<td>BM_EventPerf_EBusIncrement</td>
<td>29686 ns</td>
<td>29297 ns</td>
<td>22400</td>
</tr>
<tr>
<td>BM_EventPerf_EBusIncrementLambda</td>
<td>24516 ns</td>
<td>24554 ns</td>
<td>28000</td>
</tr>
</tbody>
</table>

**AZ::Interface<T>**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the `AZ::Interface<T>` template class to create global or application lifetime message request buses that support systems of type `T`. This template class is used to implement access for registered singletons across module boundaries. In this case, a singleton is an instance of a type that inherits `AZ::Interface::Registrar`. Once the singleton instance is registered, you can access the environment variables through code implemented on the instance. You can also make changes to the environment variables that can be viewed by other parts of your game's components.

Commonly, `AZ::Interface` should be used when you want to invoke methods on a core system like the renderer or the console from another component.

A system is an instance of a class that inherits the `Registrar` method from `AZ::Interface`. Systems that are registered with `AZ::Interface` are designed to replace global or application lifetime request buses that are currently implemented using `EBus`. There are a number of benefits to this new system, including vastly improved performance and compatibility with IDE standard code autocomplete functionality.

**Note**

Systems, in this usage, are key parts of the Lumberyard game engine. Some examples include the renderer, the console, the audio system, the input system, and the AI pathfinding system.

With `AZ::Interface`, you access these systems with this simplified syntax:

```cpp
AZ::Interface<{system-interface-here}>->Get()->PerformCommand
```

For example,

```cpp
AZ::Interface<IAudio>->Get()->PlaySound();
```

Likewise, you can use this syntax to invoke behaviors across systems for console functors (cfuncs) declared with `AZ::Console` (p. 1837).

`AZ::Interface<T>` provides a number of significant improvements over using a single handler `EBus`, such as:

- Improved performance. Calls to the singleton are a virtual function call that can often even be de-virtualized by the compiler, rather than a lock / list iterate / function dispatch to a virtual call.
- Improved debuggability. `AZ::Interface` is essentially just an `AZ::Environment` variable wrapper that enables extensible singletons within the Lumberyard engine.
AZ::Interface is defined as a C++ template (template <T>) in the following header: %INSTALL-ROOT%dev\Code\Framework\AzCore\AzCore\Interface\Interface.h

**Using AZ::Interface**

This is the process for registering a singleton thread for a system with AZ::Interface

- Obtain a raw interface pointer to a type T class instance for registration. You can assume that the registered system will outlive any cached references.
- Register the system with the interface at initialization time by calling Register() on the reference to it.
- Wait. If registration is successful, an AZ::Environment (which contains the environment variables for your game) is successfully attached and is ready to receive messages to update the environment variables.

To deregister a system, call Unregister() on AZ::Interface.

AZ::Interface defines the following static methods:

- static void Register(T* type) — Registers an instance of type T to AZ::Interface.
- static void Unregister(T* type) — Deregisters an instance of type T from AZ::Interface.
- static T* Get() — Gets a reference to an instance of type T that is registered with AZ::Interface.

It also defines a helper class, Registrar, that enables registration and deregistration from within the AZ::Interface class constructor and destructor, respectively.

```cpp
/**
 * A helper utility RAII mixin class that will register / unregister within the
 * constructor / destructor, respectively.
 *
 * Example Usage:
 * @code{.cpp}
 *      class System
 *          : public Interface<ISystem>::Registrar
 *      {
 *          
 *          @{endcode
 *      };
 * @endcode
 */

class Registrar : public T {
public:
    Registrar();
    virtual ~Registrar();
};
```

In most cases, you use Registrar instead of implementing registration directly.

Here’s an example of using AZ::Interface::Registrar to register a system with a single method, DoSomething(), defined for it.

```cpp
class ISystem {
public:
    virtual ~ISystem();
    virtual void DoSomething() = 0;
};
```
class System
    : public AZ::Interface<ISystem>::Registrar
{    
    public:
        void DoSomething() override;
};

// In client code.
// Check that the pointer is valid before use.
if (ISystem* system = AZ::Interface<ISystem>::Get())
{
    system->DoSomething();
}

Important

The restrictions for AZ::Interface are similar to that of a single-handler EBus:

- Use AZ::Interface on long-lived instances only, such as instances with global variables that live across the lifetime of a module or application.
- Because AZ::Interface uses AZ::Environment variables across DLL boundaries, you can only register/unregister after the AZ::Environment instance is attached after successful registration.
- AZ::Interface works with EBus, and you can soft-migrate EBus code by providing an AZ::Interface<T> handler for the same set of requests.
- **Thread safety is your responsibility.** Using AZ::Interface<T> does not make threads safe.

Vs. AZ::Event

AZ::Event (p. 1842) is a publish/subscribe (pub/sub) event handler that can be used when you want to subscribe to notifications from another component on the same thread. AZ::Interface, on the other hand, is a replacement for singletons, when you want to invoke methods on a core system like the renderer or the console.

Converting from an EBus implementation

Here is an example of converting a global request bus to AZ::Interface<T>.

**Example Original original EBus baseline**

```cpp
// Bus interface
class EBusRequests
    : public AZ::EBusTraits
{    
    public:
        using MutexType = NullMutex;
        static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;
        static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
        virtual void Request(int value) = 0;
};
using EBusEventExampleBus = AZ::EBus<EBusEventExample>;

// Bus implementation
class EBusEventExampleImpl
    : public EBusPerfBaselineBus::Handler
{    
    public:
```
Example EBus implementation converted to use AZ::Interface

To convert from all-in EBus usage to a global request bus that uses AZ::Interface but can still interoperate with Script Canvas, you must make a few changes:

- Create your pure virtual interface without any EBus code.
- Create an EBus wrapper that inherits from AZ::EBusTraits, and declare the EBus as AZ::EBus<{your-interface-class-name}, {your-ebus-wrapper-name}>.
- Create your implementation of your interface, inherit from your EBus wrapper handler, and do the following:
  - Call both Register() and BusConnect() in your class constructor.
  - Call both Unregister() and BusDisconnect() in your class destructor.

```cpp
// Our pure-virtual interface only
class IRequests
{
    public:
        virtual void Request(int value) = 0;
};

// EBus stuff
class EBusStuff
    : public AZ::EBusTraits
{
    public:
        using MutexType = NullMutex;
        static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;
        static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
    }
using EBusStuffBus = AZ::EBus<IRequests, EBusStuff>; // Note we specify the pure virtual interface first, and then the EBus stuff after

// Implementation, inherit from the pure-virtual interface
class RequestsImpl
    : public EBusStuffBus::Handler // Note we inherit from the bus handler
{
    public:
        AZ_RTTI(RequestsImpl, "{some guid}", IRequests); // AZ type info is required

        RequestsImpl()
        {
            AZ::Interface<IRequests>::Register(this);
            EBusStuffBus::Handler::BusConnect();
        }

        ~RequestsImpl()
        {
            EBusStuffBus::Handler::BusDisconnect();
            AZ::Interface<IRequests>::Unregister(this);
        }

        void Request(int value) override;
};
```

// Invoke a request
EBusRequestsBus::Broadcast(&EBusRequests::Request, 1);
Example

If you don't require interoperation with Script Canvas, you can avoid using EBus entirely, as seen in this example.

```cpp
// Our pure-virtual interface only
class IRequests
{
public:
    virtual void Request(int value) = 0;
};

// Implementation
class RequestsImpl : public IRequests
{
public:
    AZ_RTTI(RequestsImpl, "{some guid}", IRequests); // AZ type info is required

    RequestsImpl()
    {
        AZ::Interface<IRequests>::Register(this);
    }

    ~RequestsImpl()
    {
        AZ::Interface<IRequests>::Unregister(this);
    }

    void Request(int value) override;
};

// Invoke a request
AZ::Interface<IRequests>::Get()->Request(1);
```

Unit testing

The AZ::Interface system includes a number of unit tests to validate correct behavior.

To execute the unit tests, the following command-line arguments can be provided to the AzTestRunner:

```
%INSTALL-ROOT%\dev\Bin64vc141.Test\AzCoreTests.dll AzRunUnitTests --pause-on-completion --gtest_break_on_failure --gtest_filter=InterfaceTest*
```

You should see unit testing output like this:

```
[==========] Running 6 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 6 tests from InterfaceTest
[ RUN      ] InterfaceTest.EmptyInterfaceTest
[       OK ] InterfaceTest.EmptyInterfaceTest (2 ms)
[ RUN      ] InterfaceTest.EmptyAfterDisconnectTest
[       OK ] InterfaceTest.EmptyAfterDisconnectTest (0 ms)
[ RUN      ] InterfaceTest.ValidInterfaceTest
[       OK ] InterfaceTest.ValidInterfaceTest (0 ms)
```
Working with the Event Bus (EBus) system

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Event buses (EBuses) are a general-purpose communication system that Lumberyard uses to dispatch notifications and receive requests. EBuses are configurable and support many different use cases.

To interact with the engine or other components in Lumberyard, include the component or system's EBus or API header in your code. Then make calls to the exposed EBuses. With this approach you can replace engine–level system APIs with implementations that you define in a gem. For example, you could replace Lumberyard's audio system with your own EBus handler. This would give you complete control over audio without having to recompile the engine.

For examples of EBus usage, see Usage and Examples (p. 1853).

For in-depth information about EBuses, including conceptual diagrams, see Event Buses in Depth (p. 1857).

For C++ API reference documentation on the core EBus code, see the EBus API Reference in the Amazon Lumberyard C++ API Reference.

How Components Use EBuses

Components commonly use EBuses in two ways: to dispatch events or to handle requests. A bus that dispatches events is a notification bus. A bus that receives requests is a request bus. Some components provide one type of bus, and some components provide both types. Some components do not provide an EBus at all. You use the EBus class for both EBus types, but you configure the EBuses differently. The following sections show how to set up and configure notification buses, event handlers, and request buses.

Notification Buses

Notification buses dispatch events. The events are received by handlers, which implement a function to handle the event. Handlers first connect to the bus. When the bus dispatches an event, the handler's function executes. This section shows how to set up a notification bus to dispatch an event and a handler to receive the event.

Setting up a Notification Bus

To set up a bus to dispatch events

1. Define a class that inherits from EBusTraits. This class will be the interface for the EBus.
2. Override individual EBusTraits properties to define the behavior of your bus. Three EBusTraits that notification buses commonly override are AddressPolicy, which defines how many addresses the EBus contains, HandlerPolicy, which describes how many handlers can connect to each address, and BusIdType, which is the type of ID that is used to address the EBus if addresses are used. For example, notification buses often need to have multiple addresses, with the addresses identified by entity ID. To do so, they override the default AddressPolicy with EBusAddressPolicy::ById and set the BusIdType to EntityId.

3. Declare a function for each event that the EBus will dispatch. Handler classes will implement these functions to handle the events.

4. Declare an EBus that takes your class as a template parameter.

5. Send events. The function that you use to send the event depends on which addresses you want to send the event to, whether to return a value, the order in which to call the handlers, and whether to queue the event.

   - To send an event to all handlers connected to the EBus, use Broadcast(). If an EBus has multiple addresses, you can use Event() to send the event only to handlers connected at the specified ID. For performance-critical code, you can avoid an address lookup by using Event() variants that take a pointer instead of an ID.
   - If an event returns a value, use BroadcastResult() or EventResult() to get the result.
   - If you want handlers to receive the events in reverse order, use BroadcastReverse() or EventReverse().
   - To send events asynchronously, queue the event. Queued events are not executed until the queue is flushed. To support queuing, set the EnableEventQueue trait. To queue events, use QueueBroadcast() or QueueEvent(). To flush the event queue, use ExecuteQueuedEvents().

Setting up a Handler

To enable a handler class to handle the events dispatched by a notification bus

1. Derive your handler class from <BusName>::Handler. For example, a class that needs to handle tick requests should derive from TickRequestBus::Handler.
2. Implement the EBus interface to define how the handler class should handle the events. In the tick bus example, a handler class would implement OnTick().
3. Connect and disconnect from the bus at the appropriate places within your handler class's code. Use <BusName>::Handler::BusConnect() to connect to the bus and <BusName>::Handler::BusDisconnect() to disconnect from the bus. If the handler class is a component, connect to the bus in Activate() and disconnect from the bus in Deactivate(). Non-components typically connect in the constructor and disconnect in the destructor.

Request Buses

A request bus receives and handles requests. Typically, only one class handles requests for a request bus.

Setting up a Request Bus

The first several steps for setting up a request bus are similar to setting up a notification bus. After that you also need to implement the handlers for handling the requests.

To set up a request bus

1. Define a class that inherits from EBusTraits. This class will be the interface for requests made to the EBus.
2. Override individual EBusTraits properties to define the behavior of your bus. Two EBusTraits that request buses commonly override are AddressPolicy, which defines how many addresses the EBus contains, and HandlerPolicy, which describes how many handlers can connect to each address. For example, because there is typically only one handler class for each request bus, request buses typically override the default handler policy with EBusHandlerPolicy::Single.

3. Declare a function for each event that the handler class will receive requests about. These are the functions that other classes will use to make requests of the handler class.

4. Declare an EBus that takes your class as a template parameter.

5. Implement a handler for the events as described in the previous section Setting up a Handler (p. 1852).

Usage and Examples

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This section provides examples in C++ of how to declare and configure an EBus, implement a handler, send messages, and receive return values. For information about using EBuses in Lua, see Using EBuses in Lua (p. 2693).

Topics
- Declaring an EBus (p. 1853)
- EBus Configuration Options (p. 1854)
- Implementing a Handler (p. 1855)
- Sending Messages to an EBus (p. 1856)
- Retrieving Return Values (p. 1856)
- Return Values from Multiple Handlers (p. 1856)
- Asynchronous/Queued Buses (p. 1857)

Declaring an EBus

Declaring an EBus is much like declaring any virtual interface class in C++. However, you can specify various configuration options that control how the EBuses is generated at compile time and how it behaves.

Here is a simple example of a basic interface and associated EBuses.

```cpp
class ExampleInterface : public AZ::EBusTraits
{
public:
    // ------------------ EBus Configuration -------------------
    // These override the defaults in EBusTraits.
    // One handler per address is supported.
    static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;

    // The EBus contains a single address.
    static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
    // ------------------------ Other -------------------------
}
```
virtual ~ExampleInterface() { };

// ------------------ Handler Interface -------------------
// Handlers inherit from ExampleInterfaceBus::Handler

// Handlers are required to implement this because it's pure virtual.
virtual void DoSomething() = 0;

// Handlers can override this, but are not required to.
virtual void SomeMessage() { }

// Returns a value and has a parameter.
virtual bool ReturnsValue(int x) = 0;
}

using ExampleInterfaceBus = AZ::EBus<ExampleInterface>;

Tip

Use descriptive names in EBuses, and avoid overloaded functions. Explicit and descriptive function names prevent future API name collisions when classes inherit your EBus interfaces. Avoiding overloaded functions improves the experience of using your EBuses. This is especially true from scripting environments such as Lua, in which descriptive names improve readability and clarity. For more information, see Components and EBuses: Best Practices (p. 1026).

EBus Configuration Options

EBus configuration options are key to controlling how the EBus behaves. The configuration options used in the previous example are explained in the following sections.

HandlerPolicy

The HandlerPolicy trait determines how many handlers connect to an address on the EBus and the order in which handlers at each address receive events. The following example specifies a single handler (p. 1858):

// One handler per address is supported.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;

The HandlerPolicy has two common uses:

- A singleton pattern in which various systems post messages or requests to a single system elsewhere in the codebase.
- A pattern where a specific component or an entity handles messages to the EBus. For example, you might have a mesh component that owns an entity. The mesh component handles all mesh-related queries addressed to the entity's ID.

Address Policy

The AddressPolicy trait defines how many addresses exist on the EBus. The following example specifies only a single address. An ID is not required.

// The EBus contains a single address.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;

Practical use cases for a single address policy include any global EBus that is not tied to a specific entity, application-specific ID, or object.
**EBusAddressPolicy Options**

The `EBusAddressPolicy` has the following options:

- **Single** – The EBus uses a single address. No ID is used. The EBus can have a single handler (p. 1858) or many handlers (p. 1859).
- **ById** – The EBus has multiple addresses. The order in which addresses are notified when broadcasting events without an ID is not specified.
- **ByIdAndOrdered** – The EBus has multiple addresses. However, when broadcasting events without an ID, we want to control the order in which individual addresses are notified. The `BusIdOrderCompare` definition allows for arbitrary customization of ordering.

**EBusHandlerPolicy Options**

The `EBusHandlerPolicy` has the following options:

- **Single** – One handler per address is supported. Uses include an EBus with a single handler (p. 1858) or an EBus with addresses and a single handler (p. 1860).
- **Multiple** – Any number of handlers are supported. Ordering is ignored. Uses include many handlers (p. 1859) or an EBus with addresses and many handlers (p. 1861).
- **MultipleAndOrdered** – Any number of handlers are supported, and handlers are notified in a particular order. The `BusHandlerOrderCompare` definition allows for arbitrary customization of ordering.

**Implementing a Handler**

A handler of an EBus derives from `AZ::EBus<x>::Handler`. For convenience this was defined as `ExampleInterfaceBus` in the previous example (p. 1853). This means that the handler can be derived from `ExampleInterfaceBus::Handler`.

```cpp
#include "ExampleInterface.h"

// note: derives from bus handler, rather than directly from ExampleInterface
class MyHandler : protected ExampleInterfaceBus::Handler
{
public:
   void Activate();

protected:
   // Implement the handler interface:
   void DoSomething() override; // note: Override specified.
   void SomeMessage() override;
   bool ReturnsValue(int x) override;
};
```

Note that handlers are not automatically connected to an EBus, but are disconnected automatically because the destructor of `Handler` calls `BusDisconnect`.

In order to actually connect to the EBus and start receiving events, your handler must call `BusConnect()`:

```cpp
void MyHandler::Activate()
{
   // For a single EBus, this would be just BusConnect().
   // For multiple EBuses, you must specify the EBus to connect to:
   ExampleInterfaceBus::Handler::BusConnect();
}
You can call `BusConnect()` at any time and from any thread.

If your EBus is addressed, connect to the EBus by passing the EBus ID to `BusConnect()`.

```cpp
// connect to the EBus at address 5.
ExampleAddressBus::Handler::BusConnect(5);
```

### Sending Messages to an EBus

Anyone who can include the header can send messages to the EBus at any time. Using the previous example, a completely unrelated class can issue a `DoSomething` call on the EBus:

```cpp
// Note: You don't need to include MyHandler.h.
ExampleInterfaceBus::Broadcast(&ExampleInterfaceBus::Events::DoSomething);
```

EBuses also support a macro-based syntax. This syntax is being phased out, but uses of it can still be found in Lumberyard source code. The macro syntax for the previous example is as follows.

```cpp
#include "ExampleInterface.h" // Note: You don't need to include MyHandler.h.
...
EBUS_EVENT(ExampleInterfaceBus, DoSomething);
```

If your EBus is addressed, you can send events to a specific address ID. Events broadcast globally are received at all addresses.

```cpp
// Broadcasts to ALL HANDLERS on this EBus regardless of address (even if the EBus has addresses)
ExampleAddressBus::Broadcast(&ExampleAddressBus::Events::Test);

// Broadcasts only to handlers connected to address 5.
ExampleAddressBus::Event(5, &ExampleAddressBus::Events::Test);
```

### Retrieving Return Values

If you make a synchronous call, you can also supply a variable in which to place the result:

```cpp
// ALWAYS INITIALIZE YOUR RESULT!!!
// Since there may be nobody connected to the EBus, your result may not be populated.
bool result = false;
ExampleInterfaceBus::BroadcastResult(result, &ExampleInterfaceBus::Events::ReturnsValue, 2);
```

In this example, if there are no handlers connected to the EBus, the `result` variable is not modified. If one or more handlers are connected to the EBus, `operator=` is called on the `result` variable for each handler.

### Return Values from Multiple Handlers

In certain cases you might have to aggregate the return value of a function when there are multiple handlers. For example, suppose you want to send a message to all handlers that asks whether any one
handler objects to shutting down an application. If any one handler returns true, you should stop the shutdown. The following would not suffice:

```cpp
// Counterexample: returnValue contains only the result of the final handler.
bool returnValue = false;
SomeInterfaceBus::BroadcastResult(returnValue, &SomeInterfaceBus::Events::DoesAnyoneObject);
```

Because the EBus issues `operator=` for each handler, `returnValue` would contain only the result of the final handler.

Instead, you can create a class to collect your results that overrides `operator=`. There are several built-in types for this, and you can make your own:

```cpp
#include <AZCore/EBus/Results.h>
...
AZ::EBusAggregateResults<bool> results;
SomeInterfaceBus::BroadcastResult(results, &SomeInterfaceBus::Events::DoesAnyoneObject);
```

// results now contains a vector of all results from all handlers.

// alternative:
```cpp
AZ::EBusLogicalResult<bool, AZStd::logical_or<bool>> response(false);
SomeInterfaceBus::BroadcastResult(response, &SomeInterfaceBus::Events::DoesAnyoneObject);
```

// response now contains each result, using a logical OR operation. So all responses are OR'd with each other.

**Note**
Additional building blocks (for example, arithmetic results) are available inside the `results.h` file.

### Asynchronous/Queued Buses

To declare an EBus on which events can be queued and sent asynchronously, add the following to the EBus declaration:

```cpp
static const bool EnableEventQueue = true;
```

You can use `QueueBroadcast` and `QueueEvent` to enqueue events on an EBus so that you can flush them later from a controlled location or thread.

To flush the queue at the appropriate location or thread, invoke the following:

```cpp
ExampleInterfaceBus::ExecuteQueuedEvents();
```

### Event Buses in Depth

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Event buses (or EBus for short) are a general purpose system for dispatching messages. Ebuses have many advantages:
• **Abstraction** – Minimize hard dependencies between systems.

• **Event-driven programming** – Eliminate polling patterns for more scalable and high performing software.

• **Cleaner application code** – Safely dispatch messages without concern for what is handling them or whether they are being handled at all.

• **Concurrency** – Queue events from various threads for safe execution on another thread or for distributed system applications.

• **Predictability** – Provide support for ordering of handlers on a given bus.

• **Debugging** – Intercept messages for reporting, profiling, and introspection purposes.

You can use EBuses in many different ways. Following are some examples:

• As a direct global function call

• Dispatch processing to multiple handlers

• Queue all calls, acting like a command buffer

• As an addressable mailbox

• For imperative delivery

• For queued delivery

• Automatic marshalling of a function call into a network message or other command buffer

The EBus source code can be found in the Lumberyard directory location `<root>\dev\Code\Framework\AZCore\AZCore\EBus\EBus.h`.

**Bus Configurations**

You can configure EBuses for various usage patterns. This section presents common configurations and their applications.

**Topics**

• Single Handler (p. 1858)

• Many Handlers (p. 1859)

• EBus with Addresses and a Single Handler (p. 1860)

• EBus with Addresses and Many Handlers (p. 1861)

**Single Handler**

The simplest configuration is a many-to-one (or zero) communication bus, much like a singleton pattern.
There is at most one handler, to which any sender can dispatch events. Senders need not manually check and de-reference pointers. If no handler is connected to the bus, the event is simply ignored.

```cpp
// One handler is supported.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;

// The EBus uses a single address.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
```

### Many Handlers

Another common configuration is one in which many handlers can be present. You can use this configuration to implement observer patterns, subscriptions to system events, or general-purpose broadcasting.

![Diagram](image)

Events to the handlers can be received in defined or undefined order. You specify which one in the `HandlerPolicy` trait.

#### Example Without Handler Ordering

To handle events in no particular order, simply use the `Multiple` keyword in the `HandlerPolicy` trait, as in the following example:

```cpp
// Multiple handlers. Events received in undefined order.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Multiple;

// The EBus uses a single address.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;
```

#### Example with Handler Ordering

To handle events in a particular order, use the `MultipleAndOrdered` keyword in the `HandlerPolicy` trait, and then implement a custom handler-ordering function, as in the following example:

```cpp
// Multiple handlers. Events received in defined order.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::MultipleAndOrdered;

// The EBus uses a single address.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::Single;

// Implement a custom handler-ordering function
```
EBus with Addresses and a Single Handler

EBuses also support addressing based on a custom ID. Events addressed to an ID are received by handlers connected to that ID. If an event is broadcast without an ID, it is received by handlers at all addresses.

A common use for this approach is for communication among the components of a single entity, or between components of a separate but related entity. In this case the entity ID is the address.

Example Without Address Ordering

In the following example, messages broadcast with an ID arrive at each address in no particular order.
// One handler per address is supported.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;

// The EBus has multiple addresses. Addresses are not ordered.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::ById;

// Messages are addressed by EntityId.
using BusIdType = AZ::EntityId;

Example With Address Ordering

In the following example, messages broadcast with an ID arrive at each address in a specified order.

// One handler per address is supported.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Single;

// The EBus has multiple addresses. Addresses are ordered.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::ByIdAndOrdered;

// Messages are addressed by EntityId.
using BusIdType = AZ::EntityId;

// Addresses are ordered by EntityId.
using BusIdOrderCompare = AZStd::greater<BusIdType>;

EBus with Addresses and Many Handlers

In the previous configuration, only one handler is allowed per address. This is often desirable to enforce ownership of an EBus for a specific ID, as in the singleton case above. However, if you want more than one handler per address, you can configure the EBus accordingly:
Example: Without Address Ordering

In the following example, messages broadcast with an ID arrive at each address in no particular order. At each address, the order in which handlers receive the message is defined by EBusHandlerPolicy, which in this example is simply ById:

```cpp
// Allow any number of handlers per address.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Multiple;

// The EBus has multiple addresses. Addresses are not ordered.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::ById;

// Messages are addressed by EntityId.
```
using BusIdType = AZ::EntityId;

Example: With Address Ordering

In the following example, messages broadcast with an ID arrive at each address in a specified order. At each address, the order in which handlers receive the message is defined by the EBusHandlerPolicy, which in this example is ByIdAndOrdered.

```cpp
// Allow any number of handlers per address.
static const AZ::EBusHandlerPolicy HandlerPolicy = AZ::EBusHandlerPolicy::Multiple;

// The EBus has multiple addresses. Addresses are ordered.
static const AZ::EBusAddressPolicy AddressPolicy = AZ::EBusAddressPolicy::ByIdAndOrdered;

// We address the bus EntityId.
using BusIdType = AZ::EntityId;

// Addresses are ordered by EntityId.
using BusIdOrderCompare = AZStd::greater<BusIdType>;
```

**Synchronous vs. Asynchronous**

EBus supports both synchronous and asynchronous (queued) messaging.

**Synchronous Messaging**

Synchronous messages are sent to any and all handlers when an EBus event is invoked. Synchronous messages limit opportunities for asynchronous programming, but they offer the following benefits:

- They don't require storing a closure. Arguments are forwarded directly to callers.
- They let you retrieve an immediate result from a handler (event return value).
- They have no latency.

**Asynchronous Messaging**

Asynchronous messages have the following advantages:

- They create many more opportunities for parallelism and are more future proof.
- They support queuing messages from any thread, dispatching them on a safe thread (like the main thread, or any thread that you choose).
- The code used to write them is inherently tolerant to latency and is easily migrated to actor models and other distributed systems.
- The performance of the code that initiates events doesn't rely on the efficiency of the code that handles the events.
- In performance-critical code, asynchronous messages can improve i-cache and d-cache performance because they require fewer virtual function calls.

For information on declaring an EBus for queing and sending messages asynchronously, see Asynchronous/Queued Buses (p. 1857).

**Additional Features**

EBuses contain other features that address various patterns and use cases:
Common Tasks, EBuses, and Handlers

- **Cache a pointer to which messages can be dispatched** – This is handy for EBuses that have IDs. Instead of looking up the EBus address by ID for each event, you can use the cached pointer for faster dispatching.

- **Queue any callable function on an EBus** – When you use queued messaging, you can queue a Lambda function or bound function against an EBus for execution on another thread. This is useful for general purpose thread-safe queuing.

---

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following are some common game programming tasks and the EBuses and handlers that you can use to implement them.

**Detect Mouse, Keyboard, or Other Button Events**

<table>
<thead>
<tr>
<th>Bus</th>
<th>AZ::InputEventNotificationBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>OnPressed, OnHeld, OnReleased</td>
</tr>
<tr>
<td>File</td>
<td>InputEventBus.h</td>
</tr>
</tbody>
</table>

Use these events to detect when mouse, keyboard, or other buttons are pressed, held, or released.

**Detect Entity or Component Readiness**

<table>
<thead>
<tr>
<th>Bus</th>
<th>LmbrCentral::MeshComponentNotificationBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>OnMeshCreated, OnMeshDestroyed</td>
</tr>
<tr>
<td>File</td>
<td>MeshComponentBus.h</td>
</tr>
</tbody>
</table>

Even after an entity has been created and its components have been activated, visual data might not be fully loaded. The OnMeshCreated event occurs when the mesh creation is complete. This is useful if you want to access the underlying ICharacterInstance and ISkeletonAnim members in order to play animations. More generally, it is useful to declare a component or entity as "alive" or game ready, whatever that might mean for your application.

**Detect When a Member Joins or Leaves a Session**

<table>
<thead>
<tr>
<th>Bus</th>
<th>GridMate::SessionEventBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>OnMemberJoined, OnMemberLeaving</td>
</tr>
<tr>
<td>File</td>
<td>Session.h</td>
</tr>
</tbody>
</table>
You can use the SessionEventBus to detect when a member joins or leaves a network session. For documentation on this EBus, see Reacting to Session Events (p. 2032) in the Using Lumberyard Networking (p. 1990).

### Get and Set Physics Characteristics

<table>
<thead>
<tr>
<th>Bus</th>
<th>LmbrCentral::PhysicsComponentRequestBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>AddImpulse, GetMass, SetMass, GetVelocity, SetVelocity, etc.</td>
</tr>
<tr>
<td>File</td>
<td>PhysicsComponentBus.h</td>
</tr>
</tbody>
</table>

The PhysicsComponentRequestBus contains useful methods for getting or setting the physical characteristics of objects like mass, density, velocity, and water damping. For an example of using a pointer directly to the underlying handler for better access to functions such as GetVelocity, see Direct Access to EBus Handlers (p. 1866).

### Get Notifications for Animation Events

<table>
<thead>
<tr>
<th>Bus</th>
<th>LmbrCentral::CharacterAnimationNotificationBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>OnAnimationEvent</td>
</tr>
<tr>
<td>File</td>
<td>CharacterAnimationBus.h</td>
</tr>
</tbody>
</table>

If you have set up animations in the .animevents file in Geppetto, an OnAnimationEvent event is called for each animation event during animation playback. You can monitor this to get notifications for animation events. The string configured for the animation event in Geppetto is held in the LmbrCentral::AnimationEvent::m_animName variable.

### Get or Set the Location of an Entity in the World

<table>
<thead>
<tr>
<th>Bus</th>
<th>AZ::TransformBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>GetWorldX, SetWorldX, GetWorldY, SetWorldY, etc.</td>
</tr>
<tr>
<td>File</td>
<td>TransformBus.h</td>
</tr>
</tbody>
</table>

The TransformBus contains many useful methods for getting or setting where in the world the entity is, such as xyz axis locations. For an example of using a pointer directly to the entity’s transform for more optimal access to functions such as GetBasisY (the entity’s forward vector), see Direct Access to EBus Handlers (p. 1866).

### Manually Play Animations

<table>
<thead>
<tr>
<th>Bus</th>
<th>LmbrCentral::SkinnedMeshComponentRequestBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>GetCharacterInstance</td>
</tr>
</tbody>
</table>
To play animations manually, use the ISkeletonAnim in the character instance. To get the ISkeletonAnim from the ICharacterInstance, use ICharacterInstance::GetISkeletonAnim().

**Use an EBus from Another Component**

<table>
<thead>
<tr>
<th>Bus</th>
<th>AZ::EntityBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>OnEntityActivated, OnEntityDeactivated</td>
</tr>
<tr>
<td>File</td>
<td>EntityBus.h</td>
</tr>
</tbody>
</table>

The OnEntityActivated and OnEntityDeactivated events are called after all of an entity's components have had their Activate() or Deactivate() function called. These events can be useful if you want your component to use an EBus that another component has already set up in its Activate() function.

**Use Tick Events**

<table>
<thead>
<tr>
<th>Bus</th>
<th>AZ::TickBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>OnTick</td>
</tr>
<tr>
<td>File</td>
<td>TickBus.h</td>
</tr>
</tbody>
</table>

A tick is a unit of time generated by the component application. The OnTick event signals that the application has issued a tick and is called each frame. By default, handlers receive events based on the order in which the components are initialized, but you can override this. For more information, see Tick Bus and Components (p. 1019).

**Direct Access to EBus Handlers**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

At run time, EBuses form a communication backbone between a component and other noncomponent subsystems. In a nontrivial application, EBus activity can consume a substantial amount of CPU and memory usage. On data-cache (D-Cache)–sensitive devices such as consoles, the number of data lookups on each EBus call can potentially degrade performance. To reduce the number of data lookups, you can use a pointer to the EBus handler instead of calling the EBus.

The result of an EBus interaction is identical to a virtual function call on an EBus handler. As a result, you can implement this approach by getting a pointer directly to the handler at initialization time. Later, at run time, you make a direct call to the handler’s function. Because you know in advance whether or not a handler pointer is null, you do not have to initialize a request bus return value every time.

The code examples in this document compare the two approaches.
Using PhysicsComponentRequestBus to Get the Velocity Every Tick

The following example gets the velocity indirectly by using the physics component EBus.

class PlayerAnimationController : public AZ::Component, private AZ::TickBus::Handler
{
    public:
        // AZ::Component
        void Activate() override
        {
            AZ::TickBus::Handler::m_tickOrder = AZ::ComponentTickBus::TICK_PHYSICS + 1; // To be updated after the velocity is determined.
            AZ::TickBus::Handler::BusConnect(); // for OnTick
        }

        void Deactivate() override;
    }

    private:
        // AZ::TickBus::Handler
        void OnTick(float deltaTime, AZ::ScriptTimePoint time) override
        {
            // Get the velocity indirectly by using the physics component EBus.
            AZ::Vector3 velocity = AZ::Vector3::CreateZero();
            EBUS_EVENT_ID_RESULT(velocity, GetEntityId(), LmbrCentral::PhysicsComponentRequestBus, GetVelocity);

            // Do something with velocity here.
        }
};

The following example uses FindFirstHandler() to cache the handler and gets the velocity by calling the handler directly.

class PlayerAnimationController : public AZ::Component, private AZ::TickBus::Handler, private AZ::EntityBus::Handler
{
    public:
        // AZ::Component
        void Activate() override
        {
            AZ::EntityBus::Handler::BusConnect(GetEntityId()); // For OnEntityActivated/OnEntityDeactivated.
        }

        void Deactivate() override;
    }

    private:
        // AZ::EntityBus::Handler
        void OnEntityActivated(const AZ::EntityId &parentEntityId) override
        {
            // Do this after all the other entity's components have been activated so that the physics component has been already attached to the EBus.
            m_physicsComponentRequests = LmbrCentral::PhysicsComponentRequestBus::FindFirstHandler(GetEntityId());
            AZ::TickBus::Handler::m_tickOrder = AZ::ComponentTickBus::TICK_PHYSICS + 1; // To be updated after the velocity is determined.
            AZ::TickBus::Handler::BusConnect(); // for OnTick.
        }
};
void OnEntityDeactivated(const AZ::EntityId &parentEntityId) override
{
    AZ::TickBus::Handler::BusDisconnect();
    m_physicsComponentRequests = nullptr;
}

// AZ::TickBus::Handler
void OnTick(float deltaTime, AZ::ScriptTimePoint time) override
{
    // Get the velocity directly from the physics component handler.
    AZ::Vector3 velocity = m_physicsComponentRequests != nullptr ?
        m_physicsComponentRequests->GetVelocity() : AZ::Vector3::CreateZero();

    // Do something with velocity here.
}

LmbrCentral::PhysicsComponentRequests *m_physicsComponentRequests = nullptr;
};

Using TransformBus to Get the WorldTransform's Forward Vector Every Tick

The following example gets the world transform's forward vector by indirectly by using the transform EBus.

class PlayerAnimationController : public AZ::Component, private AZ::TickBus::Handler
{
    public:
        // AZ::Component
        void Activate() override
        {
            AZ::TickBus::Handler::m_tickOrder = AZ::ComponentTickBus::TICK_PHYSICS + 1; // //
            To be updated after the velocity is determined.
            AZ::TickBus::Handler::BusConnect(); // for OnTick
        }

    void Deactivate() override;
    {
        AZ::TickBus::Handler::BusDisconnect();
    }

    private:
        // AZ::TickBus::Handler
        void OnTick(float deltaTime, AZ::ScriptTimePoint time) override
        {
            // Get the forward vector indirectly by using the transform EBus.
            AZ::Transform worldTransform = AZ::Transform::Identity();
            EBUS_EVENT_ID_RESULT(worldTransform, GetEntityId(), AZ::TransformBus, GetWorldTM);
            AZ::Vector3 forward = worldTransform.GetBasisY();

            // Do something with "forward" here.
        }
    }
};

The following example caches the handler and calls it directly. Because every entity has a transform, the entity already has an API operation for accessing the transform. Using the EBus and the FindFirstHandler function is not necessary.

class PlayerAnimationController : public AZ::Component, private AZ::TickBus::Handler, private AZ::EntityBus::Handler
{
public:
    // AZ::Component
    void Activate() override
    {
        AZ::EntityBus::Handler::BusConnect(GetEntityId()); // for OnEntityActivated/OnEntityDeactivated
    }

    void Deactivate() override;
    {
        AZ::EntityBus::Handler::BusDisconnect(GetEntityId());
    }

private:
    // AZ::EntityBus::Handler
    void OnEntityActivated(const AZ::EntityId &parentEntityId) override
    {
        m_worldTransform = &GetEntity()->GetTransform()->GetWorldTM();
        AZ::TickBus::Handler::m_tickOrder = AZ::ComponentTickBus::TICK_PHYSICS + 1; // To be updated after the velocity is determined.
        AZ::TickBus::Handler::BusConnect(); // for OnTick
    }

    void OnEntityDeactivated(const AZ::EntityId &parentEntityId) override
    {
        AZ::TickBus::Handler::BusDisconnect();
        m_worldTransform = nullptr;
    }

    // AZ::TickBus::Handler
    void OnTick(float deltaTime, AZ::ScriptTimePoint time) override
    {
        // Get the forward vector directly from the AZ::Transform.
        AZ::Vector3 forward = m_worldTransform->GetBasisY();

        // Do something with "forward" here.
    }

    const AZ::Transform *m_worldTransform = nullptr;
};

Automating boilerplate with AZ Code Generator

AZ Code Generator is a command line utility that generates source code (or any data or text) from specially tagged source code. You can use it when the structure of the intended code is known in advance so that templates can be made for it. For example, you could generate boilerplate code for serialization or reflection.

AZ Code Generator parses a list of existing C++ source files and/or header files and generates intermediate data in JSON format. It passes the intermediate data to a series of templates.

The templates provide the format for the code that is generated. Templates make increased coding efficiency possible because they enable automatic updates of boilerplate code. When a template is
updated, all related generated code is regenerated in the next build. This removes the need to update the glue code manually or to use error-prone find-and-replace operations.

### Workflow Summary

The following steps describe how AZ Code Generator works with Waf to generate code.

1. The Waf build system invokes AZ Code Generator for the `.h` and `.cpp` source files that are specified in the `wscript` file.
2. AZ Code Generator runs one or more passes with the specified files.
3. Each pass includes the following:
   a. AZ Code Generator uses the Clang front-end compiler to produce an abstract syntax tree (AST) for each provided source file. The Clang parser attempts to compile the input. For increased speed, Clang can be instructed to not follow `#include` statements and to suppress all errors.
   b. The AST is translated into an intermediate JSON format.
   c. The intermediate JSON object is passed into a template driver as a Python script and then into a Jinja2 template. Each driver and template implements specific code generation tasks.
d. The template driver performs any preprocessing that you want on the intermediate JSON object.

e. The intermediate JSON is then passed to Jinja2 templates.

f. Each template driver can have an arbitrary number of templates, which can output to an arbitrary number of output files. Multiple templates can have the same output file or different output files as the template driver creator wants.

4. AZ Code Generator returns a list of generated files to the Waf build system.

5. The Waf build system completes the build process, including the generated code in the build.

The following sections provide more detail about this process.

**Waf**

The AZ Code Generator is fully integrated into the Using the Waf Build System (p. 63). You can use the Waf `az_code_gen` feature to invoke the AZ Code Generator. We recommend that you use Waf rather than the command line to start the `AzCodeGenerator.exe` utility.

For examples and more information about the Waf integration, see AZ Code Generator Integration with Waf (p. 1874).

**Clang**

The default front end of the AZ Code Generator is a Clang parser/compiler for C++ source code. AZ Code Generator uses Clang to parse source code (which might include custom tags) and generate the intermediate JSON data object. AZ Code Generator fully controls Clang's parser and compilation phase so that it can selectively suppress or enable features such as diagnostics. This gives AZ Code Generator the flexibility to ignore source code that might fail to compile and still attempt to generate a complete intermediate object.

**Intermediate JSON Data**

The Clang front end compiler outputs an intermediate JSON data structure that the generator passes to templates for further processing. An example intermediate JSON data object follows.

```json
[
  {
    'name': 'Component',
    'qualified_name': 'AZ::Component',
    'fields': [],
    'bases': [],
    'meta': {
      'path': 'D:\Repo\Ly\branches\AzComponents\Code\Tools\AzCodeGenerator\CodeGenTest.h'
    },
    'type': 'class',
    'annotations': {},
    'methods': []
  },
  {
    'name': 'TestingClass',
    'qualified_name': 'TheNamespace::TestingClass',
    'fields': [
      {
        'type': 'float',
        'name': 'm_field2',
        'qualified_name': 'TheNamespace::TestingClass::m_field2',
        'annotations': {}
      }
    ]
  }
]
```
AZ Code Generator and Python

AZ Code Generator depends on Python 3.7 or later to run template drivers and render Jinja templates. The Python C API is used to extend Python with methods in the azcg_extension module that permit template drivers to report dependencies, errors, and useful informational output. In Windows, Python 3.7 is included in the Lumberyard dev/Tools/Python directory. On macOS, AZ Code Generator uses the version of Python that is included with the operating system.

Note
To debug Python C API calls when using AZ Code Generator, you must download CPython. Then make a build for your intended debug OS.

Template Drivers and Template Rendering

You can use template drivers written in Python to alter the intermediate data structure before passing it to the template engine. After preprocessing, the template driver might direct the Jinja2 template engine to render one or many templates, depending on the generated code that you want.

AZ Code Generator uses the Jinja2 template engine, which is downloaded by the Python easy_install script in the \dev\Tools\Python\3.7.5\windows\Scripts directory. The engine is then copied into the Lumberyard 3rdParty\jinja2 directory. Lumberyard also provides a jinja_extensions module, which contains helper methods that you can use inside templates. These extensions are stored in the dev/Code/Tools/AzCodeGenerator/Scripts/jinja_extensions/ directory. For examples and more information about Jinja templates, see Code Generation Templates (p. 1880).

Generated Files

The following sample output was generated from a serialization template. The reference JSON object has been formatted for readability.
// THIS CODE IS AUTOGENERATED, DO NOT MODIFY

#include "stdafx.h"
#include <AZCore/serialization/serializecontext.h>
#include <AzCore/Math/Vector3.h>
#include "D:/Repo/Ly/branches/AzComponents/Code/Tools/AzCodeGenerator/CodeGenTest.h"

namespace Components
{
    void TestingClassReflect(AZ::ReflectContext* reflection)
    {
        AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(reflection);
        if (serializeContext)
        {
            serializeContext->Class<TestingClass>()->SerializerForEmptyClass();
        }
    }
}

/*
// Reference JSON object

{ 'name': 'Component',
  'qualified_name': 'AZ::Component',
  'fields': [],
  'bases': [],
  'meta': { 'path': 'D:\Repo\Ly\branches\AzComponents\Code\Tools\AzCodeGenerator\CodeGenTest.h' },
  'type': 'class',
  'annotations': {},
  'methods': [] },

{ 'name': 'TestingClass',
  'qualified_name': 'TheNamespace::TestingClass',
  'fields': [
    { 'type': 'float',
      'name': 'm_field2',
      'qualified_name': 'TheNamespace::TestingClass::m_field2',
      'annotations': {} } ],
  'bases': [
    { 'name': 'Component',
      'qualified_name': 'AZ::Component' } ],
  'meta': {
AZ Code Generator Integration with Waf

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

AZ Code Generator is fully accessible for any Waf target as the feature az_code_gen. The dev\Tools \Build\waf-<version_number>\lmbrwaflib\az_code_generator.py file contains the core of the Waf integration code. It includes the az_code_gen feature that can be used by any wscript file.

The minimum required information is a list of the files to pass into the code generator and at least one template driver. This list feeds the code generator one file at a time and invokes the templates specified by the driver. The files output from the driver are added as dependencies of the build task. Output files also have the option to be reinjected back into the C++ build for compilation. Output file paths are automatically added as include paths both for the current target build and as export_header entries.

Topics
- Basic Integration (p. 1874)
- Advanced Integration (p. 1875)
- Input Files (p. 1875)
- Template Drivers (p. 1876)
- Command Line Parameters (p. 1876)
- Waf Specific Options (p. 1876)

Basic Integration

In the wscript file for the target requiring generated code, add the az_code_gen feature as follows.
features = ['az_code_gen'],

Next, specify the files to pass as input to the code generator, as in the following example.

```python
az_code_gen = [
    {'files': ['MySourceFile.h'],
     'scripts': ['MyTemplateDriver.py']},
],
```

The paths given are relative to the target path in both cases.

Whenever the specified target is compiled, a code generation task passes in the `MySourceFile.h` file to the code generator. It also invokes the `MyTemplateDriver.py` file to control the output. For information on how to write a template driver, see Template Drivers (p. 1882).

**Advanced Integration**

The AZ Code Generator Waf integration uses passes to define the code generator tasks that must be run during build time. Each pass determines the set of files, drivers, and environment settings with which to run the code generator. Currently, all passes are run in parallel without any dependency checking between passes.

The following example shows the configuration of multiple passes.

```python
az_code_gen = [
    {'files': ['MyCode/MySourceFile.h'],
     'scripts': ['MyCode/MyTemplateDriver.py']},
    {'files': ['MyOtherCode/MyOtherSourceFile.h'],
     'scripts': ['MyOtherCode/MyOtherTemplateDriver.py']}
],
```

This example generates the following two code generation tasks.

1. Pass in the `MyCode/MySourceFile.h` file to the code generator and invoke the `MyCode/MyTemplateDriver.py` file to control the output.
2. Pass in `MyOtherCode/MyOtherSourceFile.h` to the code generator and invoke `MyOtherCode/MyOtherTemplateDriver.py` to control the output.

**Input Files**

Each pass provides a list of files that will be used as input to the code generator. This list can also contain string paths, nodes, and lists. Top-level string paths and nodes are passed individually to the code generator. Note the following:

- If you provide a list, all files or nodes in that list are used by the code generator at the same time. This allows for maximum flexibility, but typical usage is one input per task.
- The overhead of the Waf task and AZ Code Generator bootstrapping can be significant. To improve performance, you can pass in multiple input files in one list.
- The code generator invokes the same Clang and template driver pipeline for each input file.

The following example shows several input file specifications.
# Finds this file relative to the build context source node
files : [bld.srcnode.find_or_declare('Code/Framework/AzCore/Tests/CodeGen.h')],

'files' : [
# Pass both MyClass.h and MyClass.cpp at the same time to code generator to get more
# information about MyClass than just the header. Note the nested lists.
    ['MyClass.h', 'MyClass.cpp']
]

'files' : [
    # Any and all variations are allowed, but because lists provide only one layer of
    # lists are allowed only at the top level.
    'MySourceFile.h',
    'MyOtherSourceFile.cpp',
    bld.srcnode.find_or_declare('Code/Framework/AzCore/Tests/CodeGen.h'),
    ['MyClass.h', 'MyClass.cpp']
]

Template Drivers

To specify template drivers to use for each code generation pass, provide a list of string paths, relative to
the target path, as in the following example.

'scripts' : [
    '../../../Framework/AzFramework/CodeGen/AzClassCpp.py',
    '../../../Framework/AzFramework/CodeGen/AzEBusInline.py',
    '../../../Framework/AzFramework/CodeGen/AzReflectionCpp.py',
    '../../../Framework/AzFramework/CodeGen/AzClassInline.py'
],

Command Line Parameters

All command line parameters for the code generation utility can be specified in each code generation
pass. To do this, provide a list of arguments, as in the following example.

'arguments' : [
    '-OnlyRunDiagnosticsOnMainFile=true',
    '-SuppressDiagnostics=false',
    '-SuppressErrorsAsWarnings=false',
    '-output-redirection=file',
],

For a full list of parameters, see AZ Code Generator Parameters (p. 1877).

Waf Specific Options

The Waf integration provides additional options that can be specified in a list for each code generation
pass, as in the following example.

'options' : ['PrintOutputRedirectionFile'],

PrintOutputRedirectionFile – This option, when used in combination with the -output-
redirection=file parameter, directs Waf to provide AZ Code Generator a path to save extra output
during code generation. The path to this file is listed for each task during the build if errors occur.

Profile – This option enables profiler timings of clang parsing and script execution within the AZ Code
Generator tool.
AZ Code Generator Parameters

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For best results, pass the options for AZ Code Generator in to the Waf build system. However, you can also specify the parameters for AzCodeGenerator.exe on the command line.

Topics
- Waf Parameters (p. 1877)
- Clang Compilation Parameters (p. 1877)
- Intermediate Data (p. 1877)
- Front End (p. 1878)
- AZ Code Generator Parameter List (p. 1878)

Waf Parameters

Most parameters for AZ Code Generator are specified by the Waf integration. Parameters such as input, output, and include paths are automatically detected and forwarded. Other AZ Code Generator parameters control how AZ Code Generator deals with the source code input and the intermediate data that is generated.

Specify any of these in the arguments section of the az_code_gen pass in the wscript file.

Clang Compilation Parameters

The following AzCodeGenerator.exe parameters apply to Clang compilation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- SuppressIncludeNotFoundError</td>
<td>Suppresses unknown #include statements at compile time.</td>
</tr>
<tr>
<td>- OnlyRunDiagnosticsOnMainFile</td>
<td>Ignores build warnings and errors on all except the main file specified for compilation.</td>
</tr>
<tr>
<td>- SuppressDiagnostics</td>
<td>Ignores build warnings and errors on all files.</td>
</tr>
<tr>
<td>- SuppressErrorsAsWarnings</td>
<td>Downgrades any build errors to warnings. Allows Clang to succeed even if there are errors.</td>
</tr>
</tbody>
</table>

Intermediate Data

To include information about code outside of the input file in the intermediate JSON data, use the following option.

- inclusion-filter="wildcard filter for files to allow"
Front End
You can choose the front end to use by specifying either the `-Clang` (the default) or `-JSON` option.

AZ Code Generator Parameter List
The following list shows all AZ Code Generator parameters.
Usage: `AzCodeGenerator.exe [options]`

<table>
<thead>
<tr>
<th>Option</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Clang</code></td>
<td>General</td>
<td>Uses the Clang compiler front end.</td>
</tr>
<tr>
<td><code>-clang-settings-file=&lt;string&gt;</code></td>
<td>Code parsing</td>
<td>The path to the file that contains Clang configuration settings.</td>
</tr>
<tr>
<td><code>-codegen-script=&lt;string&gt;</code></td>
<td>Python</td>
<td>The absolute path and file name of the code generation script to invoke.</td>
</tr>
<tr>
<td><code>-debug</code></td>
<td>General</td>
<td>Enables debug output.</td>
</tr>
<tr>
<td><code>-debug-buffer-size=&lt;uint&gt;</code></td>
<td>General</td>
<td>Buffers the last n characters of debug output until program termination. The default is 0, which specifies immediate print out.</td>
</tr>
<tr>
<td><code>-debug-only=&lt;debug string&gt;</code></td>
<td>General</td>
<td>Enables a specific type of debug output.</td>
</tr>
<tr>
<td><code>-define=&lt;string&gt;</code></td>
<td>Code parsing</td>
<td>Specifies a preprocessor definition.</td>
</tr>
<tr>
<td><code>-DelayedTemplateParsing</code></td>
<td>AST traversal</td>
<td>Consumes and stores template tokens for parsing at the end of the translation unit.</td>
</tr>
<tr>
<td><code>-EnableIncrementalProcessing</code></td>
<td>AST traversal</td>
<td>Enables incremental processing.</td>
</tr>
<tr>
<td><code>-force-include=&lt;string&gt;</code></td>
<td>Code parsing</td>
<td>List of headers to forcibly include in Clang parsing.</td>
</tr>
<tr>
<td><code>-help</code></td>
<td>General</td>
<td>Displays basic options in categorized format.</td>
</tr>
<tr>
<td><code>-help-hidden</code></td>
<td>General</td>
<td>Displays all available options in categorized format.</td>
</tr>
<tr>
<td><code>-help-list</code></td>
<td>General</td>
<td>Displays basic options in list format.</td>
</tr>
<tr>
<td><code>-help-list-hidden</code></td>
<td>General</td>
<td>Displays all available options in list format.</td>
</tr>
<tr>
<td><code>-include-path=&lt;string&gt;</code></td>
<td>Code parsing</td>
<td>The header includes the path.</td>
</tr>
<tr>
<td><code>-inclusion-filter=&lt;string&gt;</code></td>
<td>Code filtering</td>
<td>Specifies a wildcard filter so that files other than those specified by input-files are parsed by Clang into intermediate data.</td>
</tr>
<tr>
<td><code>-info-output-file=&lt;filename&gt;</code></td>
<td>General</td>
<td>File to which to append -stats output.</td>
</tr>
<tr>
<td><code>-input-file=&lt;string&gt;</code></td>
<td>Code parsing</td>
<td>(Required) Path to input file relative to the value of input-path.</td>
</tr>
</tbody>
</table>
### AZ Code Generator Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-input-path=&lt;string&gt;</td>
<td>Code parsing</td>
<td>(Required) The absolute path to input folder. All input-file paths must be relative to this folder.</td>
</tr>
<tr>
<td>-intermediate-file=&lt;string&gt;</td>
<td>Code parsing</td>
<td>Path to a file that stores the JSON AST from Clang parsing.</td>
</tr>
<tr>
<td>-JSON</td>
<td>General</td>
<td>Uses raw JSON input for the front end.</td>
</tr>
<tr>
<td>-noscripts</td>
<td>General</td>
<td>Disables the running of code generation scripts.</td>
</tr>
<tr>
<td>OnlyRunDiagnostics</td>
<td>Clang</td>
<td>Runs diagnostics (error and warning checking) only on the main file that is compiled. Ignores errors and warnings from all other files.</td>
</tr>
<tr>
<td>-output-path=&lt;string&gt;</td>
<td>Code parsing</td>
<td>(Required) The absolute path to the output folder.</td>
</tr>
<tr>
<td>-output-redirection</td>
<td>Output</td>
<td>Redirects output and error messages from Clang and Python internal utilities.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=none – No output redirection. Clang and Python output to stdout and stderr.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=null – Redirect Clang and Python to null, effectively suppressing output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=file – Redirect Clang and Python to disk. Use redirect-output-file to specify the path.</td>
</tr>
<tr>
<td>-output-using-json</td>
<td>Output</td>
<td>Outputs using JSON objects instead of plain text. Use this option to ease parsing for calling applications.</td>
</tr>
<tr>
<td>-print-all-options</td>
<td>General</td>
<td>Prints all option values after command line parsing.</td>
</tr>
<tr>
<td>-print-options</td>
<td>General</td>
<td>Prints nondefault options after command line parsing.</td>
</tr>
<tr>
<td>-profile</td>
<td>General</td>
<td>Enables AZ Code Generator's internal profiler and emits timings for Clang parsing and script execution.</td>
</tr>
<tr>
<td>-python-debug-path=&lt;string&gt;</td>
<td>Python</td>
<td>Path to Python debug libraries and scripts for AzCodeGenerator.exe to use in debugging.</td>
</tr>
<tr>
<td>-python-home=&lt;string&gt;</td>
<td>Python</td>
<td>(Required) The equivalent of the PYTHONHOME environment variable, which is ignored.</td>
</tr>
<tr>
<td>-python-home-debug=&lt;string&gt;</td>
<td>Python</td>
<td>The equivalent of the debug Python PYTHONHOME environment variable, which is ignored.</td>
</tr>
<tr>
<td>-python-path=&lt;string&gt;</td>
<td>Python</td>
<td>The path to Python libraries and scripts for AzCodeGenerator.exe.</td>
</tr>
<tr>
<td>-redirect-output-file=&lt;string&gt;</td>
<td>Output</td>
<td>The file path for redirected output. Use in combination with the -output-redirection=file option. The default file name is output.log.</td>
</tr>
<tr>
<td>-resource-dir=&lt;string&gt;</td>
<td>Code parsing</td>
<td>The path to the resource directory for Clang.</td>
</tr>
<tr>
<td>Option</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-stats</td>
<td>General</td>
<td>Enables statistics output from program (available with asserts). Use the <code>-info-output-file=&lt;filename&gt;</code> option to specify the output file.</td>
</tr>
<tr>
<td>-SkipFunctionBodies</td>
<td>AST traversal</td>
<td>Does not traverse function bodies.</td>
</tr>
<tr>
<td>-SuppressDiagnostics</td>
<td>Clang compilation</td>
<td>Hides Clang compilation diagnostic information.</td>
</tr>
<tr>
<td>-SuppressErrorsAsWarnings</td>
<td>Clang compilation</td>
<td>Suppresses compilation errors during parsing by reporting them as warnings.</td>
</tr>
<tr>
<td>-SuppressIncludeNotFoundError</td>
<td>AST traversal</td>
<td>Suppresses #include not found errors.</td>
</tr>
<tr>
<td>-track-memory</td>
<td>General</td>
<td>Enables <code>-time-passes</code> memory tracking. Performance might be slow when this option is used.</td>
</tr>
<tr>
<td>-v</td>
<td>General</td>
<td>Outputs verbose debug information.</td>
</tr>
<tr>
<td>-version</td>
<td>General</td>
<td>Displays the version of AzCodeGenerator.exe.</td>
</tr>
<tr>
<td>-view-background</td>
<td>General</td>
<td>Executes the graph viewer in the background. This option creates a .tmp file that must be deleted manually.</td>
</tr>
</tbody>
</table>

**Code Generation Templates**

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

AZ Code Generator uses the Jinja2 template engine for Python to render its output. The Jinja template engine outputs plain text with embedded variable and logic statements.

Jinja templates are designed to be highly readable and mimic the overall structure of the desired output. They are processed top to bottom. Any text outside of the control block in the template is sent directly to the output.

The following are some example templates. For more information about creating Jinja templates, refer to the Jinja Template Designer Documentation.

**Topics**

- Simple Example (p. 1881)
- Complex Example (p. 1881)
- Template Data (p. 1882)
Simple Example

A Jinja template can use text variables to replace text at predetermined locations in the output, as in the following example:

```plaintext
// Here's a {{ variable_name }} !
int {{ variable_name }} = {{ variable_value }};
```

In this example, the Jinja template is given the following input.

```json
{
  'variable_name' = 'foo',
  'variable_value' = 42
}
```

The following output results.

```plaintext
// Here's a foo !
int foo = 42;
```

Complex Example

Jinja allows for fairly complicated logic, branching and looping control structures. The following example template generates a class that has the public and private variables specified by the input:

```plaintext
// This class is auto-generated!
class {{ class.name }}
{
public:
    virtual ~{{ class.name }}() = default;

{% if class.members is defined %}
    {% for member_var in class.members if member_var.visibility is 'public' %}
        {{ member_var.type }} m_{{ member_var.name }}{{ if member_var.value is defined }} = {{ member_var.value }}{{ endif }};
    {% endfor %}
{% endif %}

private:
{% if class.members is defined %}
    {% for member_var in class.members if member_var.visibility is 'private' %}
        {{ member_var.type }} m_{{ member_var.name }}{{ if member_var.value is defined }} = {{ member_var.value }}{{ endif }};
    {% endfor %}
{% endif %}
};
```

In this example, the Jinja template is given the following input.

```json
{
    'class': {
        'name': 'MyClass',
        'members': [
            {
                'name': 'foo',
                'type': 'int',
                'visibility': 'public'
            },
            {
                'name': 'bar',
```
The template produces the following output.

```cpp
// This class is auto-generated!
class MyClass
{
public:
    virtual ~MyClass() = default;
    int m_foo;
    long m_bar;
private:
    float m_secretSauce = 98.6f;
};
```

### Template Data

The data that is available to the template is fully controlled by the Python template driver (p. 1882).

The following table lists the variables that are automatically added to the Jinja environment.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra_data</td>
<td>Python object that contains data returned by the apply_transformations (p. 1884) method of the template driver.</td>
</tr>
<tr>
<td>extra_str</td>
<td>String that contains the contents of extra_data in JSON format.</td>
</tr>
<tr>
<td>json_object</td>
<td>Python object that contains the decoded intermediate JSON after it has been processed by the template driver.</td>
</tr>
<tr>
<td>json_str</td>
<td>String that contains the encoded intermediate JSON after it has been processed by the template driver.</td>
</tr>
</tbody>
</table>

For information about the intermediate output, see Intermediate JSON Data Format (p. 1901).

**Note**

Because Jinja contains a limited feature set, attempting to do complex data transformations in Jinja templates produces overly complicated and generally unreadable templates. For this reason, we recommend that you perform any major data manipulation in the template driver before it is passed into the Jinja template engine. For more information, see Template Drivers (p. 1882).

### Template Drivers

AZ Code Generator is in preview release and is subject to change.
Template drivers are Python scripts that process the intermediate JSON data and route it into the Jinja2 output templates. The scripts preprocess the data from the Clang front end, execute the template rendering, and control where the generated output is written to disk.

These scripts are usually called by one or more code generation passes in WAF wscript files. Each Python script can reference multiple templates. This offers great flexibility in implementation, especially when multiple templates rely on the same preprocessed data.

Topics
- Specifying Drivers in Waf (p. 1883)
- Creating a Template Driver in Python (p. 1883)
- Minimal Template Driver (p. 1885)
- Rendering Templates (p. 1885)
- Configuring Automatic Build Injection (p. 1886)
- Preprocessing Intermediate Data (p. 1886)

Specifying Drivers in Waf
Drivers are specified by file name in each code generation pass. The file path is relative to the root of the wscript target. All drivers are invoked on each input file.

The following shows the structure of a sample Waf entry.

```python
'az_code_gen' = [
    {
        'files': [ '<files to gen>' ],
        'scripts': [ '<list of script file paths relative to current wscript folder>' ]
    }
]
```

For more details on how to specify passes, see AZ Code Generator Integration with Waf (p. 1874).

Creating a Template Driver in Python
To create a template driver in Python, you must import the TemplateDriver base class and override its methods. The code for the class can be found in the dev/Code/Tools/AzCodeGenerator/Scripts/az_code_gen/base.py file.

This class is automatically injected into Python by AZ Code Generator and only needs to be imported as az_code_gen.base, as in the following example.

```python
from az_code_gen.base import *
```

Methods to Override in the TemplateDriver Class
To implement your template driver, override the following methods in the TemplateDriver class.

- add_dependency

Call the add_dependency method to manually add a dependency to the az_code_gen task in Waf. The file path given should be absolute so that the render template can specify additional dependencies that
Waf does not automatically include. These dependencies might be external data files used to render the templates, or files that were used to generate the input data.

**Syntax**

```python
add_dependency(self, dependency_file)
```

**apply_transformations**

Override the `apply_transformations` method to manipulate the raw JSON object, which is passed in as the `obj` parameter. Manipulations are performed in place on the object. The object is then forwarded through the pipeline and is eventually passed to `jinja_args` of `render_templates`. Any object returned by this method is provided to the Jinja environment as `extra_data`.

**Syntax**

```python
apply_transformations(self, obj)
```

For an example of this method, see Preprocessing Intermediate Data (p. 1886).

**get_expected_tags**

Override the `get_expected_tags` method to return a list of tags that must be found in any input file. If the required tags are not present, this driver is skipped.

**Important**

This method is deprecated as of Lumberyard v1.6. After Lumberyard v1.6, all scripts will be processed regardless of expected tags, and `get_expected_tags` will not be invoked.

**Syntax**

```python
get_expected_tags(self)
```

**render_template_to_file**

Renders a template to disk. This method also adds the value of `output_file` as a dependency of the `az_code_gen` task in Waf.

**Syntax**

```python
render_template_to_file(self, template_file, template_kwargs, output_file, should_add_to_build=False)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>template_file</code></td>
<td>Specifies the path to a template relative to the directory that contains the template driver .py file.</td>
</tr>
<tr>
<td><code>template_kwargs</code></td>
<td>Specifies a dictionary of key-value pairs to be passed to Jinja. Generally this should be treated as a passthrough variable for the <code>jinja_args</code> given to <code>render_templates</code>, but you can add additional key-value pairs.</td>
</tr>
<tr>
<td><code>output_file</code></td>
<td>Specifies the target file for the rendered Jinja output. The path is relative to the target output folder.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>should_add_to_build</td>
<td>A Boolean value that specifies whether Waf should add this file to the C++ build and linker. The default is false.</td>
</tr>
</tbody>
</table>

**render_templates**

Override `render_templates` to invoke template rendering by calling `render_template_to_file`.

**Syntax**

```python
render_templates(self, input_file, **jinja_args)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_file</td>
<td>The path relative to the input path that is used to invoke Clang.</td>
</tr>
<tr>
<td>jinja_args</td>
<td>The raw data from the intermediate JSON object after the template driver performs preprocessing on the object.</td>
</tr>
</tbody>
</table>

**Minimal Template Driver**

The minimum code required for a template driver is to derive from the `TemplateDriver` base class and implement a factory function to construct the template driver.

```python
from az_code_gen.base import *

class MyTemplateDriver(TemplateDriver):
    pass

# Factory function - called from launcher
def create_drivers(env):
    return [MyTemplateDriver(env)]
```

The `az_code_gen` module is automatically provided by AZ Code Generator. It contains the `TemplateDriver` and other useful methods from the `base.py` file.

The `create_drivers` function simply forwards the Jinja environment that is used to render templates. However, you can alter the function to perform other work when the driver is instantiated.

**Note**

The above bare-bones implementation works but does not generate any output.

**Rendering Templates**

To generate some output, you must implement the `render_templates` method, as in the following example.

```python
from az_code_gen.base import *

class MyTemplateDriver(TemplateDriver):
    def render_templates(self, input_file, **jinja_args):
```
```
self.render_template_to_file("MyTemplate.tpl", jinja_args, 'GeneratedCode.cpp')
```

# Factory function - called from launcher
def create_drivers(env):
    return [MyTemplateDriver(env)]

The `render_templates` method takes the relative `input_file` path and any arguments that were passed in from the `AZCodeGenerator.exe` utility. The `input_file` path usually contains inputs such as the intermediate `json_object` created by Clang.

Template drivers can extend this information by implementing the `apply_transformations` method. For more information, see Preprocessing Intermediate Data (p. 1886).

The `render_template_to_file` method takes a template file and argument key-value pairs to pass into the template engine directly and an output path to write the template engine render output to disk.

### Configuring Automatic Build Injection

At this point, the example generates a minimal `.cpp` file. The example above does not compile or link the `.cpp` file. This is appropriate if you intend to include the generated code manually using an `#include` in another file.

To inject the generated file automatically, add the `should_add_to_build` parameter to the `render_template_to_file` method and pass the parameter the value of `true`. The `should_add_to_build` parameter informs Waf that the generated file needs to be built and linked into the current target.

**Note**

Using the `should_add_to_build` parameter is not recommended for header files or other generated files that are not C++ code that must be compiled and linked.

The following example shows some build injected output.

```
from az_code_gen.base import *

class MyTemplateDriver(TemplateDriver):
    def render_templates(self, input_file, **jinja_args):
        self.render_template_to_file("MyTemplate.tpl", jinja_args, 'GeneratedCode.cpp',
                                      should_add_to_build=True)

# Factory function - called from launcher
def create_drivers(env):
    return [MyTemplateDriver(env)]
```

### Preprocessing Intermediate Data

Some cases require preprocessing of the intermediate data for easier consumption by the template engine. To do this, implement the `apply_transformations` method in your template driver. You can use this method to access the intermediate JSON data object directly before it gets passed to `render_templates`. An example follows.

```
from az_code_gen.base import *

class MyTemplateDriver(TemplateDriver):
    def render_templates(self, input_file, **jinja_args):
        self.render_template_to_file("MyTemplate.tpl", jinja_args, 'GeneratedCode.cpp')
```
def apply_transformations(self, obj):
    obj['my_custom_data'] = 42

# Factory function - called from launcher
def create_drivers(env):
    return [MyTemplateDriver(env)]

For information on the contents of the obj variable, see Intermediate JSON Data Format (p. 1901).

Custom Code Generator Annotations

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can provide additional data to your template driver by attaching annotations and tags to your source code.

Topics
- Reference Annotations (p. 1887)
- Helper Macros (p. 1887)
- Example Annotations (p. 1888)

Reference Annotations

When you create custom code generator annotations, it is a good idea to refer for examples to the existing annotations in the dev/Code/Framework/AZCore/AZCore/Preprocessor/CodeGen.h file. The existing annotations use macros extensively as a workaround for the lack of proper annotations in C++.

Clang provides an annotate attribute that can be read at parse time. You can use the helper macros provided to create new annotations, as in the following example.

```python
__attribute__((annotate("<Some string here>")))
```

This attribute is wrapped with a macro that converts its contents into strings that can be parsed by the AZ Code Generator utility.

Helper Macros

AZ Code Generator has two helper macros for annotations: AZCG_CreateAnnotation and AZCG_CreateArgumentAnnotation.

AZCG_CreateAnnotation

AZCG_CreateAnnotation is the core macro that exposes the underlying Clang annotate attribute. The macro definition follows.

```python
// AZCG_CreateAnnotation
```
#define AZCG_CreateAnnotation(annotation) __attribute__((annotate(annotation)))

Any argument passed to AZCG_CreateAnnotation must be a string.

**AZCG_CreateArgumentAnnotation**

The AZCG_CreateArgumentAnnotation macro is commonly used for annotation macros. The macro definition follows.

```c
// AZCG_CreateArgumentAnnotation
#define AZCG_CreateArgumentAnnotation AZCG_CreateAnnotation(AZ_STRINGIZE(annotation_name) "(" AZ_STRINGIZE((__VA_ARGS__)) ")")
```

The AZCG_CreateArgumentAnnotation macro takes an annotation_name argument and a number of variable arguments. The values passed to the variable arguments are collapsed into a single string for parsing by the AZ Code Generator.

**Example Annotations**

This section provides example annotations. One example forwards arguments to the underlying macro, one places an annotation inside a class, and one injects code back into the originating file.

**Simple Annotation**

The following example creates a new annotation called AzExample that forwards its arguments to the underlying macro.

```c
//Sample Annotation
#define AzExample(...) AZCG_CreateArgumentAnnotation(AzExample, __VA_ARGS__)
```

In this example, the private and public names of the annotation are the same. However, the external and internal names do not have to match.

You can attach the AzExample annotation to most items in C++, as in the following example.

```c
// Sample Tag Usage
class ExampleClass
{
    AzExample(description("I am data!"))
    int m_myData;
}
```

The tags inside the annotation are placed in JSON format in the generated intermediate data object, as in the following example. Some data has been removed for readability.

```c
// Sample Tag JSON
{
    "type": "class",
    "name": " ExampleClass",
    "annotations": {},
    "fields": [
        {
            "name": "m_myData",
            "annotations": {
                "description": "I am data!"
            }
        }
    ]
}
```
Class Annotation Example

The following example directs the AZ Code Generator utility to attach a free-floating annotation to a class.

```
// Class Tag Macro
#define AzExampleClass(...) AZCG_CreateArgumentAnnotation(AzExampleClass, Class_Attribute, __VA_ARGS__) int AZ_JOIN(m_azCodeGenInternal, __COUNTER__);
```

AzExampleClass – Specifies the annotation name AzExampleClass (instead of AzExample, as in the previous example).

Class_Attribute – Causes the AZ Code Generator utility to attach the attribute to the class that contains the annotation. The annotation belongs to the annotations property of the class object.

__VA_ARGS__ – Specifies additional parameters that are converted into a single string and passed into the AZ Code Generator utility for parsing.

```
int AZ_JOIN(m_azCodeGenInternal, __COUNTER__) -- AZ_JOIN is a helper macro that takes two macro-level entries and joins them together without converting them to strings. Because Clang requires annotation attributes be attached to a function or variable, this example uses AZ_JOIN and a temporary integer member variable to do this. The temporary integer member variable is then ignored.
```

Adding the new tag to the previous example produces the following code:

```
//Class Tag Example
class ExampleClass
{
    AzExampleClass(MyExampleClassTags::description("I am an example class!"));
    AzExample(MyExamplePropertyTags::description("I am data!"))
    int m_myData;
}
```

This produces the following intermediate JSON object. Some data has been removed for ease of comprehension.

```
// Class Tag JSON
"type": "class",
"name": "SampleClass",
"annotations": {
    "MyExampleClassTags::description": "I am an example class!"
},
"fields": [
    {
        "name": "m_myData",
        "annotations": {
            "MyExamplePropertyTags::description": "I am data!"
        }
    }
]
```

Notice that the above JSON does not look exactly like the JSON in the intermediate files provided as part of AZ framework. This is because Lumberyard uses namespaces on its tags to also provide a hierarchy for the tags on its drivers and templates. We recommend that you import the `clang_cpp.py` file and run the `format_cpp_annotations(json_object)` function on the intermediate JSON. When you do, you can use all of the convenient patterns and functions in our drivers and scripts.
The following example shows the same intermediate JSON object after processing by format_cpp_annotations().

```json
// Output of format_cpp_annotations()
"type": "class",
"name": "SampleClass",
"annotations": {
  "MyExampleClassTags": {
    "description": "I am an example class!"
  },
  "fields": [
    {"name": "m_myData",
     "annotations": {
      "MyExamplePropertyTags": {
        "description": "I am data!"
      }
    }
  ]
}
```

Generated Code Injection Example

The following example shows how to automatically inject generated code back into the original file. The example extends the previously created AzExampleClass annotation by injecting code into the example class.

```cpp
// Code Injection Macro
#if defined(AZ_CODE_GENERATOR)
  #define AzExampleClass(ClassName, ...) AZCG_CreateArgumentAnnotation(AzExampleClass,
    Class_Attribute, identifier(ClassName), __VA_ARGS__) int AZ_JOIN(m_azCodeGenInternal,
    __COUNTER__);
#else
  #define AzExampleClass(ClassName, ...) AZ_JOIN(AZ_GENERATED_CODE_,ClassName)
#endif // AZ_CODE_GENERATOR
```

The updated annotation adds a new required parameter called ClassName, which is an identifier that is used to inject the code. The identifier is passed in to Clang as identifier(ClassName), and the data is provided to the intermediate JSON.

Up until this point, the annotation macro outside of AZ_CODE_GENERATOR has been blank. The next step is to have it expand to the identifier of the code-generated macro. This causes the generated code to replace the macro annotation when the generated file is put in an #include statement.

To implement this, the example sets the macro to become AZ_JOIN(AZ_GENERATED_,ClassName). As before, AZ_JOIN in this example renders this as AZ_GENERATED_CODE_EzExampleClass. The ClassName parameter provides a name at compile time for the generated macro.

**Note**
It is not required that ClassName be the actual name of the class where the tag is used. Other tags that use this mechanism can simply require any unique identifier.

When the previous example code is updated, the following code is produced:

```cpp
// Generated Injection Code
class ExampleClass
{
  AzExampleClass(ExampleClass, description("I am an example class!")));
  AzExample(description("I am data!")));
  int m_myData;
```
This code produces the following intermediate JSON. Note the new identifier annotation on the class. Some data has been removed for readability.

```json
// Generated Code Injection JSON
"type": "class",
"name": "SampleClass",
"annotations": {
  "AzExampleClass": {
    "identifier": "ExampleClass",
    "description": "I am an example class!"
  }
},
"fields": [
  {
    "name": "m_myData",
    "annotations": {
      "AzExample": {
        "description": "I am data!"
      }
    }
  }
]
```

This result doesn't compile until the following template code used with the annotation produces the intended macro.

```c++
// Template Code
{% if class.annotations.identifier is defined %}
#define AZ_GENERATED_CODE_{{ asStringIdentifier(class.annotations.identifier) }}
public:
{# This method is injected for all classes with the AzExampleClass tag #}
bool IsExampleClass(void) { return true; }
{% endif %}
```

This code generates the following code for injection:

```c++
// Generated Code for Injection
#define AZ_GENERATED_CODE_ExampleClass
bool IsExampleClass(void) { return true; }
```

If the generated header is placed in an `#include` statement in the original code, any code in this macro will be injected into `ExampleClass`.

## Waf Debugging with AZ Code Generator

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can debug the integration output of Waf's Python scripts by using PyCharm and a few key debugging entry points. For more information about Waf integration itself, see AZ Code Generator Integration with Waf (p. 1874).
Topics

- Prerequisites (p. 1892)
- Identifying and Configuring Debug Output (p. 1892)
- Setting Up PyCharm for Debugging Waf (p. 1892)

Prerequisites

Before you start, follow the instructions for Setting Up PyCharm for Debugging Waf (p. 1892). The PyCharm debugger must be set up to debug \lmb\_waf before you can continue.

Identifying and Configuring Debug Output

All AZ Code Generator Waf integration output is prefixed with \texttt{az\_code\_gen}. To see additional output from both task creation and task execution, add \	exttt{--zones=az\_code\_gen} to the Waf command line. This exposes the commands that invoke AZ Code Generator and are useful for debugging the AZ Code Generator utility itself. For more information, see Debugging the AZ Code Generator Utility (p. 1899).

Debugging Wscript Configuration

To debug most configuration problems, it is best to set a breakpoint in the \texttt{create\_code\_generator\_tasks} method in \texttt{dev/Tools/Build/waf-<version>/\_\_waf/az\_code\_generator.py}. This method is called for each \texttt{wscript} file that uses the \texttt{az\_code\_gen} feature. It directly interprets the given passes and generates an \texttt{az\_code\_gen} task for each input file in each pass.

Debugging \texttt{az\_code\_gen} Task Creation

The \texttt{create\_az\_code\_generator\_task} feature creates \texttt{az\_code\_gen} tasks. It gathers most information and inserts it into the task. Each task gets added to the \texttt{az\_code\_gen\_group} Waf task to ensure it is executed prior to other tasks.

Debugging \texttt{az\_code\_gen} Task Execution

The \texttt{run} and \texttt{handle\_code\_generator\_output} commands are important points in task execution.

The \texttt{run} command takes the available information and generates a Clang-style arguments file prefixed with the @ symbol. The arguments file is passed on the command line to the AZ Code Generator utility.

\texttt{handle\_code\_generator\_output} - The AZ Code Generator utility returns a JSON object with one or more entries that are parsed by \texttt{handle\_code\_generator\_output}. If the AZ Code Generator utility returns an invalid, non-JSON response due to errors during execution, the Waf task returns the error message \texttt{No JSON-Object could be decoded}. To discover the return value that could not be handled, run the command outside of Waf.

Setting Up PyCharm for Debugging Waf

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Version 1.28
1892
PyCharm is an integrated development environment for Python which includes a graphical debugger that is useful for debugging Waf.

**To set up PyCharm and Waf for debugging**

2. Start PyCharm.
3. At the welcome screen, choose **Open Directory**.
4. Navigate to the `lumberyard_version\dev` directory. There should be a file called `wscript` and `waf_branch_spec.py` under this folder.
5. Configure the Python interpreter.
   a. Choose **File**, **Settings**, **Project:dev**, **Project Interpreter** to open the project interpreter page.
   b. Click the gear icon on the right of the **Project Interpreter** and choose **Add**...
   c. Select the **Existing environment** radio button, and then click the ellipse (…) icon to the right of it.
   d. Navigate to the folder containing the python executable. The executable will be contained under the path `lumberyard_version\dev\Tools\Python\3.7.5\platform`, with the exact path depending on your platform. The Python executable file must be in the same folder as the project or you may have issues running Waf.
6. Set up a debugging profile for Waf.
   
a. To set up Waf for debugging, use the project explorer in the left pane. If you don't see the project explorer, press Alt+1. Navigate to the `\dev\Tools\Build\waf-<version>` node and expand it. You should see a file called `lmbr_waf` inside this node.
b. Right-click **lmbr_waf** and choose **Create lmbr_waf**

**Note**

The **Indexing...** operation must finish before the option appears. You can verify status in the bar at the bottom.
c. In the **Create Run/Debug Configuration** dialog, ensure that the following values are configured correctly:

- **Allow parallel run** should be unselected.
- **Parameters** is the command to use to run Waf for the run/debug session.
- **Python Interpreter** should be the interpreter that you specified earlier.
- The **Working directory** must be the *dev* directory.
7. Set up `wscript` files as debuggable Python files. Waf uses files called `wscript` to define the build rules per project. These are dynamically loaded Python modules that can be debugged like any other Python module.

   a. Choose **File**, **Settings**, **Editor**, **File Types**.
b. Choose Python in Recognized File Types. Select the + icon to the right of the Registered patterns section to add a new file type.

c. Under Registered Patterns, click the green plus sign (+).

d. In the Add Wildcard dialog box, enter wscript.

8. Turn off IncrediBuild.
   a. Open the _WAF_/usersettings.options file.
   b. Verify that use_incredibuild = False

9. (Optional) Enable file outlining. This feature makes it easier to navigate source files.

   To enable file outlining, right-click the Project tab and choose Show Members.

Template Driver Debugging

AZ Code Generator is in preview release and is subject to change.
Debugging the AZ Code Generator Utility

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Because template drivers are run from the AZ Code Generator executable using Python, you can't debug them directly. However, you can debug your driver and template code (and even Jinja2 itself) by using the `debug.py` file included with AZ Code Generator.

**To debug a template driver with a Python debugger like PyCharm or Visual Studio**

1. Set the debugger to execute the `Bin64\azcg\debug.py` file. This file launches the utility to generate input JSON and emulates a code-generation pass in Python so that you can debug as if you were attached to the utility.
2. Set the working directory to `Bin64\azcg`.
3. Enter the arguments for `AzCodeGenerator.exe` into a file with one argument per line. Or use a Waf-generated arguments file as described in Waf Debugging with AZ Code Generator (p. 1891).
4. Set the arguments file, prefixed with `@`, as the argument to the script.

The following arguments are required:

- `-codegen-script` – Absolute path to the driver script that you want to debug.
- `-input-path` – Absolute path on which source file paths are based. Usually this path is the same as the location of the wscript for a given target.
- `-input-file` – Relative path from input path to the source file that is used for input.
- `-output-file` – Absolute path where generated code will be written.

After you have completed the preceding steps, you should be able to launch your debugger and set breakpoints in your driver script.

For complete AZ Code Generator parameter information, see AZ Code Generator Parameters (p. 1877).

**Debugging the AZ Code Generator Utility**

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When using Waf and the AZ Code Generator utility, you might need to debug Waf Python scripts (p. 1891) and your template drivers (p. 1898). You can also debug the AZ Code Generator utility itself, although it is less likely to be necessary. You can debug the AZ Code Generator utility by using Visual Studio in Windows or Xcode in macOS.

**Topics**

- Prerequisites (p. 1900)
- Debugging the AZ Code Generator Utility from the Waf build (p. 1900)
- Setting Visual Studio Debug Arguments (p. 1900)
- Setting Xcode Debug Arguments (p. 1901)
Prerequisites

The required preliminary steps depend on your operating system.

Windows Debugging

To debug AZ Code Generator using Visual Studio in Windows, you must generate a Visual Studio HostTools solution (.sln) file.

To generate a Visual Studio HostTools solution file

1. Enter the following command from the dev folder.

```
lmbr_waf.bat configure --enabled-game-projects= --specs-to-include-in-project-generation=host_tools --visual-studio-solution-name=HostTools
```


macOS Debugging

To enable Waf support for Xcode, perform the following steps to generate an Xcode project.

To generate an Xcode project

1. Open the dev/_WAF_/specs/all.json file.
2. Temporarily add AzCodeGenerator to modules.
3. Run ./lmbr_waf.sh configure to regenerate the Xcode project.
4. Open the dev/Solutions/LumberyardSDK.xcodeproj file.

Debugging the AZ Code Generator Utility from the Waf build

To debug the AZ Code Generator Utility from the Waf build, you must find the arguments file generated by Waf.

Waf generates an arguments file that is passed to AZ Code Generator as a command line parameter. All command line parameters from Waf for AZ Code Generator are contained inside the arguments file. This file is useful for debugging specific Waf AZ Code Generator invocations. To make the arguments file that you use available to Waf, add the --zones=az_code_gen option to the Waf command line.

Setting Visual Studio Debug Arguments

To set up debugging of AZ Code Generator from Visual Studio, perform the following steps.

To debug AZ Code Generator from Visual Studio

1. Perform the steps to set up Windows debugging as described in Prerequisites (p. 1900).
2. In the Visual Studio Solution Explorer, right-click AzCodeGenerator, and then select Properties.
3. Under Debugging, paste the path to the arguments file into Command Arguments.
4. Click OK to close the Property window.
5. Right-click AzCodeGenerator and then click Set as StartUp Project.
6. Press F5 to launch the debugger.
Setting Xcode Debug Arguments

To set up debugging of AZ Code Generator from Xcode, perform the following steps.

To debug AZ Code Generator from Xcode
1. Perform the steps to set up macOS debugging as described in Prerequisites (p. 1900).
2. In Xcode, under the Product, Scheme menu, choose AzCodeGenerator.
3. At the bottom of the Product, Scheme menu, choose Edit Scheme.
4. Under Arguments, add a new entry to Arguments Passed On Launch that contains your debug arguments.
5. Under Info, from the Executable drop down, select Other.
   a. Navigate to the directory dev/BinMac64.Debug/azcg/AzCodeGenerator.
   b. Click Choose.
6. Close the scheme editor.
7. Choose Product, Run to launch the debugger.

Intermediate JSON Data Format

AZ Code Generator is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
Download O3DE or visit the AWS Game Tech blog to learn more.

The following JSON shows the intermediate data format consumed by Jinja2 custom templates.

```json
{
    "meta": {
        "path": "<Path/To/Code/Generator/Input/File.ext>"
    },
    "objects": [
        {
            "name": "<Name of class/struct>",
            "qualified_name": "<Fully qualified name of class or struct>",
            "fields": [
                {
                    "type": "<member variable type>",
                    "canonical_type": "<member variable canonical type>",
                    "name": "<member variable name>",
                    "qualified_name": "<fully qualified member variable name>",
                    "annotations": {
                        "<annotation name>": "<annotation variable name (can be empty string)>"
                    }
                },
                ...
            ]
        }
    ]
}
```
"traits": {
  "isAbstract": <true if abstract class, false if concrete>,
  "isPOD": <true of plain old data type; otherwise, false>,
  "isPolymorphic": <true if polymorphic type; otherwise, false>
},
"bases": [
  {
    "name": "<Base Class Name>",
    "qualified_name": "<Fully qualified name of base class>"
  },
  ...
],
"meta": {
  "path": "<Path/To/File/Containing/This/Object.ext>"
},
"type": "class" or "struct",
"annotations": {
  "<annotation name>": {
    "<annotation variable name>": "<annotation variable value (can be empty string)>",
    ...
  },
  ...
},
"methods": [
  {
    "name": "<method_name>",
    "qualified_name": "<Fully qualified name of method>",
    "is_virtual": <true if virtual method; otherwise, false>,
    "annotations": {
      "<annotation name>": {
        "<annotation variable name>": "<annotation variable value (can be empty string)>",
        ...
      },
      ...
    },
    "access": "<Access level of method, one of: public, private, protected>",
    "params": [
      {
        "type": "<parameter type>",
        "canonical_type": "<parameter canonical type>",
        "name": "<parameter name>"
      },
      ...
    ],
    "uses_override": <true if override keyword is present; otherwise, false>,
    "return_type": "<return type of method>"
  },
  ...
],
...
Use the information in this section to learn about game input in Amazon Lumberyard.

Note
For information on the Input component that replaces legacy action maps, see Input (p. 642).

Input is what differentiates interactive experiences from all other entertainment media. Regardless of genre or operating system or device, every game is driven by some form of player interaction with a physical input device. There are many different types of input devices, each of which can produce a wide range of input data. Furthermore, the way this data is delivered to an application is rarely consistent across devices or operating systems. As a cross-operating system and device game engine, Lumberyard's goal is to provide independent access to input data from any supported device. The goal is to provide a common interface with these qualities:

- OS and device agnostic
- Extensible
- Efficient

In addition, input data should have these attributes:

- Must be obtainable by subscribing to input device events
- Must be obtainable on demand by polling the state or value of an input device
- Can be a custom payload, but adds no overhead when not used

Lumberyard's input system satisfies all of these requirements and replaces the deprecated input system.

AZ Framework Input

The Lumberyard input interface is called AZ framework input. AZ framework already provides abstracted interfaces for OS-specific features like file I/O and application lifecycle management, so the AZ framework code location is a natural fit.

The AZ framework input interface uses the AZCore EBus system and defines the following classes and EBuses:

- InputDevice – A base class that represents a physical input device.
- InputDeviceId – A device name and device index that together uniquely identify an InputDevice.
- InputDeviceNotificationBus – An EBus interface that subscribes to events from input devices when they connect or disconnect.
- InputDeviceRequestBus – An EBus interface that queries input devices about their associated input channels or current connected state.
- InputChannel – A base class that represents a specific source of input data (for example, the left mouse button). Input devices typically have multiple input channels.
- InputChannelId – A name that uniquely identifies an InputChannel.
- InputChannelNotificationBus – An EBus interface that subscribes to events from input channels when the channels are active or when their state or value changes.
- InputChannelRequestBus – An EBus interface that obtains an input channel from the input channel's ID. You can then query the input channel directly for its current state or value.
- InputChannelEventListener – An event monitor that inherits from InputChannelNotificationBus but provides additional features. Subscribers can use the additional features to receive events in priority order or filter events based on the source device or channel. Subscribers can also consume events so that the events aren't passed on to lower priority monitors.
**Processing the Input**

To process input, you can simply inherit from `AzFramework::InputChannelEventListener`. You can also create a filter to receive events only from specific input devices or channels, and then override `OnInputChannelEvent` to process the input data.

If you want to query the current input state directly, use `AzFramework::InputChannelRequestBus` to obtain an input channel's ID. Then query the input channel for its current state or value.

**Text Input Interfaces**

The following interfaces are designed to process text input. The input is delivered as a complete string of UTF-8 code points. This eliminates the requirement to keep track of and interpret individual code units or convert from other encodings.

- `InputTextEventNotificationBus` – An EBus interface that subscribes to text events from input devices or input channels.
- `InputTextEntryRequestBus` – An EBus interface that sends text entry requests. The requests inform input devices that the user is expecting to start or stop entering text.
- `InputTextEventListener` – An event monitor that inherits from `InputTextNotificationBus` but provides additional features. Subscribers can use the additional features to receive events in priority order or consume events so that the events aren’t passed on to lower priority monitors.

**Auxiliary Input Interfaces**

The following AZ framework input auxiliary interfaces are only implemented by certain input devices. You can use these interfaces to query or post data related to device activity like vibration effects, motion sensors, or the mouse cursor.

- `InputHapticFeedbackBus` – An EBus interface that sends haptic feedback requests to connected input devices.
- `InputMotionSensorRequestBus` – An EBus interface that sends motion sensor requests to connected input devices.
- `InputSystemCursorRequestBus` – An EBus interface that queries or changes the state, position, or appearance of the system cursor.

**Input Devices**

All input device classes inherit from `AzFramework::InputDevice`. While multiple instances of the same class can exist, every input device is identified by a unique `InputDeviceId`.

**Input Devices Included in Lumberyard**

The base class is designed so that you can inherit from it to implement new types of input devices. However, Lumberyard already includes implementations for the following devices:

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Supported Operating Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse</td>
<td>Windows, macOS</td>
</tr>
<tr>
<td>Keyboard</td>
<td>Windows and macOS</td>
</tr>
<tr>
<td>Gamepad</td>
<td>Windows, macOS, iOS, Android, consoles</td>
</tr>
</tbody>
</table>
This core set of input devices are managed by the `AzFramework::InputSystemComponent`. You can use this component to configure the type and number of input devices that your application creates at startup.

**Creating Input Devices**

You can create and implement new types of input devices by inheriting from `AzFramework::InputDevice` and creating or destroying instances of the new class as required. For examples, see the Configuring your Project for Virtual Reality (p. 3249), which define and create new types of input devices that inherit from `AzFramework::InputDevice`. Their code is located at the Lumberyard directory locations `\dev\Gems\Oculus` and `\dev\Gems\OpenVR`, respectively.

The implementation details of each input device differ depending on the device type and operating system. However, almost all devices follow a similar pattern. They use an OS-specific API to obtain raw input data for each frame and then update all the device's associated input channels accordingly.

When you implement an input device, use only your device's `TickInput` function to update the value of the input channels. This ensures that input events are delivered at the same time every frame.

Depending on how the input device gets the raw, operating system-specific input, you might have to use one of the following techniques:

- Polling the underlying input API on each call to your input device's `TickInput` function. For examples, see all `InputDeviceGamepad` implementations in the directory `\dev\Code\Framework\AzFramework\Input\Devices\Gamepad`.
- Queuing or coalescing input data from the application's main message loop until the next call to `TickInput`. For an example, see `InputDeviceMouse` in the directory `\dev\Code\Framework\AzFramework\Input\Devices\Mouse`.
- Ensuring input channels are updated in a thread-safe manner if no raw input is received on the main thread. For an example, see `InputDeviceTouchAndroid` in the directory `\dev\Code\Framework\AzFramework\Input\Devices\Touch`.

**Input Channels**

Input devices can have multiple input channels. Each channel represents a discrete source of input data that is uniquely identified by an `InputChannelId`.

Input channels use `AzFramework::InputChannelNotificationBus::OnInputChannelEvent` to broadcast events. A channel broadcasts events when the channel's state or value changes from one frame to the next or if the channel is still active or held. Input channels can also be accessed directly and polled at any time to query their current state and value.

**Creating Input Channels**

Like the `AzFramework::InputDevice` base class, you can inherit from `AzFramework::InputChannel` to implement new types of input data sources.
Input Channels Included in Lumberyard

Lumberyard provides the following input channel implementations that are used by the devices listed in the Input Devices (p. 1904) section. You can also use these implementations for new input devices.

<table>
<thead>
<tr>
<th>Input Channel</th>
<th>Example Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputChannelAnalog</td>
<td>Gamepad trigger</td>
</tr>
<tr>
<td>InputChannelAnalogWithPosition</td>
<td>Touch with pressure at a position</td>
</tr>
<tr>
<td>InputChannelAxis1D</td>
<td>Gamepad thumbstick x or y</td>
</tr>
<tr>
<td>InputChannelAxis2D</td>
<td>Gamepad thumbstick x and y together</td>
</tr>
<tr>
<td>InputChannelAxis3D</td>
<td>Motion sensor acceleration, rotation, or magnetic field</td>
</tr>
<tr>
<td>InputChannelDelta</td>
<td>Mouse wheel</td>
</tr>
<tr>
<td>InputChannelDeltaWithSharedPosition</td>
<td>Mouse movement</td>
</tr>
<tr>
<td>InputChannelDigital</td>
<td>Gamepad button or keyboard key</td>
</tr>
<tr>
<td>InputChannelDigitalWithPosition</td>
<td>Touch without pressure at a position</td>
</tr>
<tr>
<td>InputChannelDigitalWithSharedPosition</td>
<td>Mouse button at a position</td>
</tr>
<tr>
<td>InputChannelQuaternion</td>
<td>Motion sensor orientation</td>
</tr>
</tbody>
</table>

Using Memory Allocators in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard’s memory management system determines how memory is allocated. In Lumberyard version 1.16, the memory management system has been refactored. All memory allocations go through one pipeline, and memory allocation can be tracked. This makes it easier and quicker to pinpoint memory leaks or optimize memory usage to improve game performance. This improvement is especially important for mobile and console applications, where memory resources are usually more constrained than in PC environments.

Lumberyard supports all the best known memory allocation schemes. You can use Lumberyard’s allocators to categorize allocations or keep similar allocations together to improve locality or reduce fragmentation.

**Note**
For best C++ practices for managing memory in Lumberyard, see Memory Management (p. 29).

**Topics**
- Manually Allocating Memory (p. 1907)
- AZ Memory Allocators (p. 1908)
- Applying Allocators to Your Classes (p. 1910)
- AZ Allocator Schemas (p. 1910)
- Creating an Allocator (p. 1911)
## Manually Allocating Memory

Lumberyard uses the following memory allocation functions. You can find the source code in the `lumberyard_version\dev\Code\Framework\AzCore\AzCore\Memory\` directory.

<table>
<thead>
<tr>
<th>Lumberyard Function</th>
<th>Replaces</th>
<th>Overloads and Descriptions</th>
</tr>
</thead>
</table>
| `aznew`             | `operator new` | Use instead of `new T()`. If `AZCORE_ENABLE_MEMORY_TRACKING` is defined, uses the bound allocator and record tracking information for `T`.  
**Note**  
There is no `azdelete` function. The built-in `operator delete` is sufficient. |
| `azmalloc`          | `malloc` | `azmalloc(size, [alignment])`  
Uses the `SystemAllocator` to allocate memory.  
`azmalloc(size, alignment, Allocator, [allocationName])`  
Uses the specified allocator to allocate memory. `allocationName` specifies a optional name for the allocation. |
| `azcalloc`          | `calloc` | `azcalloc(size, [alignment])`  
Uses `SystemAllocator` to allocate memory and fills the memory with zeroes.  
`azcalloc(size, alignment, Allocator, [allocationName])`  
Uses the specified allocator to allocate memory and fills the memory with zeroes. Optionally specifies a name for the allocation. |
| `azrealloc`         | `realloc` | `azrealloc(ptr, size)`  
Reallocates from `SystemAllocator`.  
`azrealloc(ptr, size, Allocator, [allocationName])`  
Reallocates from the specified allocator. Optionally specifies a name for the allocation. |
| `azfree`            | `free`   | `azfree(ptr)` |
### AZ Memory Allocators

The following diagram illustrates the hierarchy of AZ memory allocators.

<table>
<thead>
<tr>
<th>Lumberyard Function</th>
<th>Replaces</th>
<th>Overloads and Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>azcreate new with allocator</td>
<td>azcreate(T, [params])</td>
<td>Uses the SystemAllocator to allocate memory and passes optional parameters wrapped in () or {} to the constructor.</td>
</tr>
<tr>
<td>azcreate new with allocator</td>
<td>azcreate(T, params, Allocator, [allocationName])</td>
<td>Uses the specified allocator to allocate memory. Optionally specifies a name for the allocation.</td>
</tr>
<tr>
<td>azdestroy delete with allocator</td>
<td>azdestroy(T*)</td>
<td>Destroys memory. azdestroy should be used only with a corresponding azcreate function.</td>
</tr>
<tr>
<td>azdestroy delete with allocator</td>
<td>azdestroy(T*, Allocator)</td>
<td>Calls the destructor, frees memory, and returns the memory to the SystemAllocator.</td>
</tr>
<tr>
<td>azdestroy delete with allocator</td>
<td>azdestroy(T*, Allocator)</td>
<td>Calls the destructor, frees memory, returns the memory to the specified allocator.</td>
</tr>
</tbody>
</table>
- **OSAllocator** – Acts as the interface to operating system memory and should be used for direct operating system allocations on the C heap. OSAllocator is booted as early as possible in `main()`, and removed last, right before returning. If you don't create OSAllocator, the SystemAllocator creates it when needed.

OSAllocator uses system calls to allocate memory. The calls are not recorded or tracked. Other allocators use OSAllocator to obtain memory from the operating system. Drillers and memory tracking tools can use OSAllocator for data debugging.

- **BestFitExternalMapAllocator** – Uses external maps to store memory tracking information for uncached memory.

- **SystemAllocator** – The system allocator is the general purpose allocator for the AZ memory library. Like all other allocators, SystemAllocator is a singleton, but it must be initialized first and destroyed last. All other allocators use SystemAllocator for internal allocations.

- **LegacyAllocator** – Handles legacy memory allocations. For more information, see Legacy Memory Management (p. 1914).

- **PoolAllocator** – Performs extremely fast small object memory allocations. PoolAllocator can allocate sizes in a range specified by `m_minAllocationSize` to `m_maxPoolSize`.

  **Note**
  
  PoolAllocator is not thread safe. If you need a thread-safe version, use ThreadPoolAllocator, or inherit from ThreadPoolBase and then write custom code to handle the synchronization.

- **ThreadPoolAllocator** – Thread safe pool allocator. If you want to create your own thread pool heap, inherit from ThreadPoolBase, as Lumberyard requires a unique static variable for the allocator type.
Applying Allocators to Your Classes

To apply an allocator to your class, use the `AZ_CLASS_ALLOCATOR` macro in your class or directly call `AZ::AllocatorInstance<some_allocator>`. AZCore relies on `AZ_CLASS_ALLOCATOR` to specify the default allocator for the class or on explicit `azcreate` and `azdestroy` calls that specify the allocator in their signature.

- If your class does not implement `AZ_CLASS_ALLOCATOR` and you call `new` or `delete`, `new` or `delete` calls use the global operator `new` or `operator delete`.
- If your class does not implement `AZ_CLASS_ALLOCATOR` and you call `aznew`, you must implement a new operator that uses the `aznew` call signature.

AZ Allocator Schemas

Each allocator commonly implements the `IAllocator` interface and uses a schema to implement the allocation algorithms and bookkeeping. This strategy enables the same schema to be used in multiple allocators.

Allocators Schemas

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AZ::HphaSchema</code></td>
<td>This is the preferred schema. It combines a small block allocator for small allocations and a red-black tree for large allocations. This provides good general purpose performance. Use this schema if you're not sure which one to use.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td><code>HphaSchema</code> is based on Dimitar Lazarov's &quot;High Performance Heap Allocator&quot; (Game Programming Gems 7, Charles River Media, 2008, pp. 15–23).</td>
</tr>
<tr>
<td><code>AZ::HeapSchema</code></td>
<td>Uses <code>nedmalloc</code> internally. Because <code>nedmalloc</code> uses thread caches to accelerate the re-use of memory, <code>HeapSchema</code> can be useful for intensive allocation processing across multiple threads.</td>
</tr>
<tr>
<td><code>AZ::BestFitExternalSchema</code></td>
<td>A best-fit allocation scheme that uses an external map to store bookkeeping outside the memory being managed. Because the tracking node is stored outside the main chunk, Lumberyard can use this allocator with uncached memory. This is most useful for GPU resource management (for example, for textures, constant buffers, and compute buffers).</td>
</tr>
<tr>
<td><code>AZ::ChildAllocatorSchema</code></td>
<td>Acts as a pass-through schema to another allocator. Use this schema to create a new allocator based on an existing allocator like <code>SystemAllocator</code>. To properly tag the memory that each gem or logical subsystem allocates, each gem or subsystem can create its own child allocator. For more information, see Creating an Allocator (p. 1911).</td>
</tr>
<tr>
<td><code>AZ::PoolSchema</code></td>
<td>A specialized schema that implements a small block allocator for managing small, high-throughput allocations. Objects are typically pooled at the cost of using more memory.</td>
</tr>
</tbody>
</table>
Creating an Allocator

We recommend that each Lumberyard gem or logical subsystem create a ChildAllocator to properly tag the memory that it allocates. This practice makes it easier to budget resource usage and get a holistic view of it.

If you choose to write your own schema, be aware that caching significant chunks of memory can be problematic. Such caching can hamper the ability of other systems to evolve to fit the content in your game. Unless you have specific requirements, we recommend that you create a ChildAllocator that eventually uses the SystemAllocator. Using a ChildAllocator ensures that your memory is as recoverable and reusable as possible.

Prior to Lumberyard version 1.16, the most common mechanism for creating a new allocator was to inherit from SystemAllocator. This practice, which creates a completely separate free list of memory usage, results in the problem of memory being spread among disparate caches. In most cases, it is better to use the inheritance ChildAllocator<SystemAllocator>, which also makes it trivial to swap the base class of your custom allocator.

To create an allocator

1. Choose a schema to use, write a custom schema, or choose an existing allocator that you want to modify. For more information, see AZ Allocator Schemas (p. 1910).
2. Inherit from AllocatorBase<your_schema> to create your Allocator class.
3. Add AZ_TYPE_INFO so that AllocatorInstance<> can properly manage your type.

Using Your Own Allocators from Containers

To use your own allocator from a container, wrap your allocator in AZ::AZStdAlloc, like the following example.

```cpp
AZStd::vector<MyClass, AZ::AZStdAlloc<CustomAllocator>>
```

Child Allocator Example

The following code example adds a custom allocator for the Script Canvas gem (p. 1201).

Example

```cpp
// Declaration of a child allocator for the ScriptCanvas module.
```
namespace ScriptCanvas
{
    class ScriptCanvasAllocator
        : public AZ::AllocatorBase<AZ::ChildAllocatorSchema<AZ::SystemAllocator>>
    {
        public:

            AZ_TYPE_INFO(ScriptCanvasAllocator, "{2C6478E2-3B0D-4DFF-88E2-ABCB3F10B96E}");
            using Schema = AZ::ChildAllocatorSchema<AZ::SystemAllocator>;
            using Base = AZ::AllocatorBase<Schema>;
            using Descriptor = Base::Descriptor;

            ScriptCanvasAllocator()
                : Base("Script Canvas Allocator", "Child Allocator used to track Script Canvas allocations")
            {
                m_schema = new (&m_schemaStorage) Schema(Descriptor());
            }

            pointer_type Allocate(size_type byteSize, size_type alignment, int flags, const char* name, const char* fileName, int lineNum, unsigned int suppressStackRecord) override
            {
                return Base::Allocate(byteSize, alignment, flags, name, fileName, lineNum, suppressStackRecord);
            }

            pointer_type ReAllocate(pointer_type ptr, size_type newSize, size_type newAlignment) override
            {
                return Base::ReAllocate(ptr, newSize, newAlignment);
            }
        
        // Alias for using ScriptCanvasAllocator with std container types.
        using ScriptCanvasAZStdAlloc = AZ::AZStdAlloc<ScriptCanvasAllocator>;
    } // namespace ScriptCanvas

    // Specialize the AllocatorInstance for ScriptCanvas to provide the allocator stored within the ScriptCanvas gem.
    // The allocator is stored with the ScriptCanvas module and therefore its lifetime lasts as long as the
    // ScriptCanvas module is loaded.
    namespace AZ
    {
        template<>
        class AllocatorInstance<ScriptCanvas::ScriptCanvasAllocator>
        {
            public:

                using AllocatorType = ScriptCanvas::ScriptCanvasAllocator;
                using Descriptor = typename AllocatorType::Descriptor;
                friend class ScriptCanvas::ScriptCanvasModule;

                static AllocatorType& Get()
                {
                    if(!s_scriptCanvasAllocatorRef)
                    {
                        if (AZ::Environment::IsReady())
                        {
                            s_scriptCanvasAllocatorRef = AZ::Environment::FindVariable<AllocatorType>(AZ::AzTypeInfo<AllocatorType>::Name());
                            AZ_Error("ScriptCanvas", s_scriptCanvasAllocatorRef, "ScriptCanvasModule has not been loaded yet");
                        }
                    }
                    return *s_scriptCanvasAllocatorRef;
                }
        }
    } // namespace AZ
In the following code example, the ScriptCanvas module creates and owns the child allocator.

Example

```cpp
// ScriptCanvas module creates and owns the child allocator.
namespace ScriptCanvas
{
    // Stores an environment variable within the ScriptCanvas Module
    static AZ::EnvironmentVariable<ScriptCanvas::ScriptCanvasAllocator> s_scriptCanvasAllocator;

    // Create ComponentDescriptors and add them to the list.
    // The descriptors will be registered at the appropriate time.
    // The descriptors will be destroyed (and thus unregistered) at the appropriate time.
    ScriptCanvasModule::ScriptCanvasModule()
    {
        ScriptCanvasAllocator::Descriptor allocatorDescriptor;
        s_scriptCanvasAllocator = AZ::Environment::CreateVariable<ScriptCanvas::ScriptCanvasAllocator>(AZ::AzTypeInfo<ScriptCanvas::ScriptCanvasAllocator>::Name());
        s_scriptCanvasAllocator->Create(allocatorDescriptor);
    }

    ScriptCanvasModule::~ScriptCanvasModule()
    {
        s_scriptCanvasAllocator->Destroy();
    }

    AZ::ComponentTypeList ScriptCanvasModule::GetRequiredSystemComponents() const
    {
        return GetCommonSystemComponents();
    }
}
```
Static Initialization

In a monolithic build, at static initialization time (before the allocators are bootstrapped), allocations are routed directly to the underlying operating system. These static allocations are tracked in a fixed size set and sent back to the OS when they are freed. They are also reported separately to memory tracking in the Global category. To discover the memory that is being allocated globally, set a breakpoint in AZ::Internal::GlobalAlloc.

Legacy Memory Management

Starting in Lumberyard version 1.16, all Cry* allocation routines route to AZ::LegacyAllocator, which you can find in the lumberyard_version\dev\Code\CryEngine\CryCommon\LegacyAllocator.h file. LegacyAllocator has the same lifetime as OSAcclorator and obtains its memory from OSAcclorator.

Cry dynamic-link libraries override the new and delete functions instead of tagging their classes with allocators. This behavior is controlled with the USE_CRY_NEW_AND_DELETE macro (lumberyard_version\dev\Code\CryEngine\CryCommon\CryMemoryManager_impl.h). This practice should not be used outside Cry DLLs. When compiled monolithically, the LegacyAllocator catches any uses of global new or delete. This allows all allocations to be tracked and managed.

Note
As of Lumberyard version 1.16, operator new and operator delete overrides are restricted to Cry DLLs.

All Cry static functions that allocate memory have been removed or wrapped in StaticInstance<T>, which creates the functions only when the functions are first accessed. StaticInstance<T> can be used in any DLL which depends on CryCommon and includes the lumberyard_version\dev\Code\CryEngine\CryCommon\platform_impl.h file.

Note
Within AZ code, there is a NewAndDelete.inl file in the lumberyard_version\dev\Code\Framework\AzCore\AzCore\Memory\ directory. You can use this code within a gem DLL, but only in nonmonolithic mode. We do not recommend using this code except to track down untagged classes or allocations.

Code Example: Automatic Unreflection of Module Classes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To automatically unreflect a module's reflected classes from the SerializeContext when the module unloads, you can create a class that contains an instance of a memory allocator separate from the SerializeContext.

To manage memory allocations of reflected GenericClassInfo classes within the module until the module is unloaded, use the class in a static variable context. Because static variable destructors are invoked when the module unloads, the destructor for the static variable can use the opportunity to unreflect all of the module's reflected GenericClassInfo classes.

This is a safety measure that can prevent shutdown errors. For example, if a gem that is no longer loaded has a reflected class, and you attempt to use SerializeContext to serialize that class, a shutdown error can occur.
Example DLL Cleanup Class

In the following code example, the `PerModuleGenericClassInfo` class tracks module-specific reflections of `GenericClassInfo` for each `SerializeContext` that is registered with the module (.dll file).

```cpp
// DLL Cleanup Class
class SerializeContext::PerModuleGenericClassInfo final
{
    public:
        PerModuleGenericClassInfo();
        ~PerModuleGenericClassInfo();

        /// Creates GenericClassInfo and registers it with the current module if it
        /// has not already been registered.
        /// Returns a pointer to the class that was created by deriving from GenericClassInfo.
        /// template <typename T>
        typename SerializeGenericTypeInfo<T>::ClassInfoType* CreateGenericClassInfo();

    private:
        /// Creates a local OSAllocator which will be used by the data members to
        /// allocate memory from the operating system heap.
        AZ::OSAllocator m_moduleOSAllocator;

        /// Creates a type alias to associative containers with a custom allocator interface.
        /// AZ::AZStdIAllocator wraps an IAllocator interface allocator and dynamically
        /// associates an allocator with a type.
        /// AZStd::unordered_map<AZ::Uuid, AZ::GenericClassInfo*,
        /// AZStd::hash<AZ::Uuid>, AZStd::equal_to<AZ::Uuid>, AZ::AZStdIAllocator>
        /// AZStd::unordered_set<SerializeContext*,
        /// AZStd::hash<SerializeContext*>, AZStd::equal_to<SerializeContext*>, AZ::AZStdIAllocator>
        GenericInfoModuleMap m_moduleLocalGenericClassInfos;
        SerializeContextSet m_serializeContextSet;
    
    // Initializes the OSAllocator and constructs the associative containers with OSAllocator.
    SerializeContext::PerModuleGenericClassInfo::PerModuleGenericClassInfo()
        : m_moduleLocalGenericClassInfos(AZ::AZStdIAllocator(&m_moduleOSAllocator))
        , mSerializeContextSet(AZ::AZStdIAllocator(&m_moduleOSAllocator))
    {
    }

    /// Cleans up all GenericClassInfo objects created within the current .dll and
    /// unregisters them from the SerializeContext.
    SerializeContext::PerModuleGenericClassInfo::~PerModuleGenericClassInfo()
    {
        // Cleans up the memory for the GenericClassInfo objects.
        for (const AZStd::pair<AZ::Uuid, AZ::GenericClassInfo*>& moduleGenericClassInfoPair :
             genericClassInfoContainer)
        {
            GenericClassInfo* genericClassInfo = moduleGenericClassInfoPair.second;
            // Explicitly invokes the destructor and clears the memory from the OSAllocator
            module.
            genericClassInfo->~GenericClassInfo();
            m_moduleOSAllocator.DeAllocate(genericClassInfo);
        }

        // Reconstructs the associative containers with the OSAllocator so that the previously
        // allocated memory is cleared. This ensures that the associative containers do not
        // attempt to deallocate memory within their destructors.
        m_moduleLocalGenericClassInfos =
            GenericInfoModuleMap(AZ::AZStdIAllocator(&m_moduleOSAllocator));
        m_serializeContextSet =
            SerializeContextSet(AZ::AZStdIAllocator(&m_moduleOS Allocator));
    }
```
HPHA Memory Debugging

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In Lumberyard 1.16 and later versions, the HPHA memory allocator provides memory debugging features to detect and trace common memory issues.

Enabling HPHA Memory Debugging

To avoid performance issues, debugging features are disabled by default.

To enable HPHA memory debugging

1. In the lumberyard_version\dev\Code\Framework\AzCore\AzCore\Memory\HphaSchema.cpp file, uncomment the following line:

```cpp
#define DEBUG_ALLOCATOR
```

2. Save the file.

3. Perform a build in debug mode. For more information, see Building Lumberyard projects (p. 61).
Characteristics and Limitations

Because of certain limitations, the HPHA debugger can help find memory issues but cannot guarantee their absence. When using HPHA memory debugging features, note the following:

- For the HPHA debugger to work, allocations must use the HPHA allocator. HPHA memory debugging does not cover allocations created by other allocators such as a PoolAllocator.
- Most of the HPHA memory debugging features assert when they detect a memory issue. When possible, the debugger prints a stack trace that indicates where the allocation happened. The stack trace is printed into the debugger output (not to the log) so that Visual Studio can recognize it. This makes it possible to double-click a trace and navigate directly to the corresponding file and line number.
- The HPHA memory debugger does not currently cover the following memory issues:
  - Buffer underflows.
  - "Far" buffer overflows. When detecting buffer overflows, Lumberyard detects changes up to 16 bytes after the memory block. If a buffer overflow writes on byte 17, Lumberyard does not detect it.
  - Lumberyard shuts down by terminating the application rather than by destroying objects. Because Lumberyard relies on the operating system to recover memory, it cannot detect issues related to shutdowns or the destruction of objects. To reproduce, isolate, and debug such memory issues, we recommend that you use unit tests.

How Memory Debugging Works

Some memory debugging features detect memory issues when an allocation is freed, and others detect issues when the HPHA allocator is destroyed. The memory debugging works by keeping a set of debug records for each allocation. When memory is requested or returned, the debugger compares the allocation or deallocation operation with the debug records. When anomalies are detected, the debugger enforces rules with asserts. The following sections describe the asserts that occur for the different memory operations.

Allocations

For memory allocation operations, the debugger performs the following tasks:

- If a previous allocation has the same pointer, the debugger asserts and prints the stack trace of the previous allocation. This usually occurs when a process overwrites the memory for the allocator's tracking structures. Because the allocator uses memory near the blocks that it allocates, a memory overflow or underflow in a neighboring block can overwrite the memory that the HPHA uses for memory tracking. When this occurs, the HPHA might consider a used block of memory to be "unused".
- Fills the memory with a quiet NaN (qNaN) pattern (0xFF, 0xC0, 0xC0, 0xFF). This is useful for detecting specific patterns of use in uninitialized memory and can detect most (but not all) cases. For more information about the qNaN pattern, see Deallocations (p. 1917).

Deallocations

For memory deallocation operations, the debugger performs the following tasks:

- Asserts if the debug record is not found. This can happen because of double deallocations or the deallocation of an invalid pointer.
- Asserts if a guard is invalid. When an allocation occurs, an additional 16 bytes ("the guard") are placed at the end of the allocation. For example, if the request is for 40 bytes, 56 bytes are assigned and 16 are used for the guard. Memory debugging assigns random values to the 16 bytes and places them in the debug record. When the deallocation happens, the 16 bytes are checked against the 16 bytes...
Overrun Detection

stored in the debug record. If they mismatch, the debugger asserts. This assert usually indicates a memory overflow (that is, an attempt to write beyond the requested size).

**Note**
This check cannot detect the cases in which the overflow writes the exact same random bytes or writes beyond the 16 byte guard.

- Asserts if the freed size does not match the allocation size. During allocation, the requested size is stored in the debug record. If the same size is not freed, a problem occurred during the deallocation.
- Refills the freed memory with the qNaN pattern. This makes it easier to detect memory accesses after the memory has been deallocated. Without this feature, the memory contents are usually available until some code reuses the memory. Filling the freed memory with the qNaN pattern helps detect this anomalous usage early.

Reallocations

Reallocations use a new block or an existing block depending on whether contiguous memory is available.

Reallocation to a New Block

When contiguous memory is not available, memory is reallocated to a new block. The debugger performs the following tasks:

- Asserts if the previous allocation is not found. Normally, the previous allocation still exists. The allocator creates a new allocation with a new memory address and then copies the contents of the previous allocation to the new allocation. If the pointer to the previous allocation is not in the debug records, the debugger asserts.
- Asserts if the guard of the previous allocation is invalid. For information on guards, see Deallocations (p. 1917).
- Asserts if a previous allocation has the same address as the new allocation. For more information, see Allocations (p. 1917).
- Fills the memory for the new allocation with the qNaN pattern. The previous block is copied over. The remaining unused part of the new allocation should have the qNaN pattern.

Reallocation to an Existing Block

When contiguous memory is available, the pointer to an existing block is used. The debugger performs the following tasks:

- Asserts if the allocation is not found.
- Asserts if the guard of the previous allocation is invalid. For information on guards, see Deallocations (p. 1917).
- Updates the debug record stack.
- Because the size has changed, writes a new guard.

Overrun Detection

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Overrun detection is an experimental feature available starting in Lumberyard version 1.21. It helps you detect corrupted memory at the time that the corruption occurs. If you think memory corruption might be caused by read or write operations outside of allocated memory, overrun detection can help you detect the problem.

**Note**
Overrun detection mode is similar to the Microsoft Debugging Tools for Windows GFlags with full page heap verification. However, it can be used with the Lumberyard memory allocators and does not require recompiling.

### Prerequisites and Limitations

The following are the prerequisites and limitations for using the Lumberyard overrun detection feature:

- You must build and run your project for Windows PC in a Debug or Profile build.
- Overrun detection is available only on platforms that use the Windows API.
- Only allocations that go through the Lumberyard allocators (p. 1906) benefit from overrun detection. Allocations that go through `new` or `malloc` do not use overrun detection unless you overload those functions to use the Lumberyard allocators.

### When to Enable Overrun Detection

The most common indicator of overwritten memory is a crash that occurs with no obvious explanation. The crash frequently occurs in a low-level system or structure, such as an AZStd:: container, or within the memory allocator. The memory overrun is not an out-of-memory error.

### Enabling Overrun Detection

Overrun detection is enabled by a setting in your project's `Game.xml` file.

**To enable overrun detection**

1. Open your project's `lumberyard_version\dev\project_name\Config\Game.xml` file.
2. Change `useOverrunDetection` from the default `false` to `true`, as shown in the following example.

```xml
<Class name="bool" field="useOverrunDetection" value="true" type="{A0CA880C-AFE4-43CB-926C-59AC48496112}"/>
```

### Using Overrun Detection

When overrun detection is enabled, debug your game as usual. Keep the following points in mind:

- Your game runs more slowly and uses significantly more memory with overrun detection enabled.
- If a system reads or writes outside allocated memory, the game crashes with a call stack at the point of the invalid read or write. An invalid read or write includes the usual `Exception thrown: invalid read/write` message near the end of the output. If this message does not appear, the exception is not a memory read or write bug.
- If the game doesn't crash, but locks up instead, you can pause the debugger to see where the game stopped.

**Note**
The detector doesn't always release memory after the memory is acquired from the operating system. Depending on the game, the detector can increase memory...
consumption as gameplay continues. If you run out of memory when you use the detector, a crash occurs in either WindowsPlatformAllocator::ReserveBytes or WindowsPlatformAllocator::CommitBytes.

Source Code Location

For the source code, see lumberyard_version\dev\Code\Framework\AzCore\AzCore\Memory\OverrunDetectionAllocator.*

Profiling, Testing, and Debugging Game Projects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes a number of tools that are used for testing builds, profiling performance, and debugging various issues that may be encountered.

Topics

• Profiler (p. 1920)
• Using AZ Test Scanner (p. 1958)
• Writing Tests for AzTest (p. 1961)
• Debugging Issues (p. 1966)
• Tracking File Access (p. 1972)

Profiler

Profiler is in preview release and is subject to change.

Profiler is a Lumberyard tool that can capture, save, and analyze network, CPU, and VRAM usage statistics. You can used the saved data to analyze network usage frame by frame, fix problems in the use of network bandwidth, and optimize the performance of your game.

To capture data, Profiler works with GridHub. When you launch Profiler, GridHub launches automatically as a supporting background process. For more information about GridHub, see Using GridHub (p. 1954).

Topics

• Profiler Tutorial (p. 1921)
• Creating and Using Annotations (p. 1934)
• Using Profiler for Networking (p. 1937)
• Using the Profiler for CPU Usage (p. 1947)
• Using Profiler for VRAM (p. 1951)
• Using GridHub (p. 1954)
Profiler Tutorial

Profiler is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can register an application in GridHub and use Profiler to capture, inspect, play back, and export the data that you collect.

Topics

- Registering Your Application (p. 1921)
- Launching Profiler (p. 1921)
- Capturing Data (p. 1921)
- Inspecting Data (p. 1923)
- Playing Back Data (p. 1927)
- Exporting Data (p. 1933)

Registering Your Application

To enable Profiler to capture information from your application, you must first register the application in GridHub. To do so, add `AzFramework::TargetManagementComponent` to the application's `SystemComponent`.

Note: Lumberyard's built-in applications already have this component added by default.

Launching Profiler

Unlike many Lumberyard utilities, you launch Profiler from its own executable file.

To launch profiler

- From the Lumberyard `dev\Bin64\` directory, run `Profiler.exe`.

Capturing Data

Profiler has two main modes of use: capture mode and inspection mode.
To use capture mode, perform the following steps.

**To capture data**

1. Click Target.

![Target: None](image1)

Profiler shows you the applications that are available for profiling:

![Target: None](image2)

2. Select a target application.

After you have selected a target, the target selector shows the state of the connection to the target. The next time you launch Profiler, it automatically selects your target for you, if it's available.

![Profiler interface](image3)

The window is divided horizontally into channels that have associated Profiler instances. A channel is a collection of Profiler instances that relate to a specific system.

3. Each Profiler instance in a channel has a unique color. A Profiler instance is active when its color is solid:

![Profiler instance](image4)

Click the color next to a Profiler instance. The color is no longer solid, showing that the Profiler instance is inactive:
Click the color again to turn on the display and activate the instance again.

4. After you have selected a target and chosen the Profiler instances that you want to see, click **Capture**.

After the capture begins, data begins to populate the channels.

5. To stop the data capture, click **Stop Capture**.

6. When prompted, save the captured data to disk. Profiler saves the data in a binary format file with a `.drl` extension, reloads the data from disk, and switches to inspection mode.

   **Note**

   If you do not save the data, it will be discarded.

**Inspecting Data**

You can use profiler to examine the data that you have captured.

**To inspect captured data**

1. In Profiler, click **File**, **Open Data**, or press **Ctrl+O**:
2. Navigate to the `.drl` file that contains your saved data and open it.

The main screen of the Profiler provides an overview of the channels of system information. This example uses a file that has 1162 frames of data:

You can use this main view to discover anomalies across channels, or to examine particular areas of interest at a high level.

When you open the main window, the scroll box at the bottom is on the right because the playback stopped at the end of the captured data.

Notice the red vertical line on the right.

3. Click in the channels area of the window.
The red vertical line moves to where you clicked. The frame indicator shows the new position of the red line. You can place the red line, which is called the scrubber, on any frame that you want to examine in detail. For finer control over the position of the scrubber, you can enter a number in the Frame indicator.

The scrubber moves accordingly.

4. To view detailed information about a frame on which the scrubber rests, click the Detailed Profiling Information icon next to the profiler instance whose data you would like to see:

Profiler instance information appears in a detail window.
Individual profilers present details in different ways, so their detail windows can look different. For information on system-specific detail windows in Profiler, see Using Profiler for Networking (p. 1937), Using the Profiler for CPU Usage (p. 1947), and Using Profiler for VRAM (p. 1951).

5. To return to capture mode from inspection mode, click the **LIVE** tab.
Playing Back Data

You can mark and play back a subset of your captured data.

Notice that after you moved the scrubber the first time, a yellow vertical line appeared on the right at the end of the data:

This yellow marker is movable and marks the end of your desired playback range. By default, it is at the end of the captured data but may be obscured by the red scrubber.
1. Scroll the window all the way to the left, to the beginning of the capture range.

![Profiler interface](image)

Now a yellow marker also appears at the beginning of the data. You can use these two yellow markers, which by default are at the beginning and end of the capture range, to restrict the range of playback to an area of data that you are interested in. You will use these shortly.

If you have many frames of data (as in this example), the initial view does not show you all frames by default.

2. To see all frames at once, click the **Frame Count Selector**, which determines the number of frames visible, and choose **All frames**:
Now you can see the entire range of captured data, with the yellow markers at the beginning and at the end:
3. Drag the two yellow markers to an area of data that you want to play back. You can ignore the position of the scrubber for now.
4. Click **Play** to start the playback:

As the data plays back, the scrubber moves from the first yellow marker to the second, and then loops back to the first.

Here are some tips to keep in mind:

- If the playback speed is too fast (the default is 60), use the **Playback Speed** option to adjust it from 1 through 60.
- If you click a location in the playback window during playback, the playback stops and moves the scrubber to the location that you clicked.
- You can place the scrubber on a frame that you are interested in and click the detail button for a profiler instance to see the detail window for the frame.
- For greater convenience and visibility, leave the profiler instance detail window open to see the data change in the detail window as the scrubber loops between markers.
5. Click **Stop** to stop the playback.

**Exporting Data**

Some Profiler instances have an export option that you can use to save data to a *.csv* file.

**To export data from a Profiler instance to a *.csv* file**

1. Click the **Save to CSV** icon for the Profiler instance whose data you want to save:

   ![Save to CSV icon](image)

   **Note**
   
   Not all profilers have the data export option.

2. To choose the fields that you want to export, click **Customize** in the export dialog box:

   ![Customize dialog box](image)
Creating and Using Annotations

Profiler is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In Profiler, annotations are a convenient way of highlighting per-frame log information from the data captured from your application. After you learn how annotations are used in Profiler, you can modify your application so that they appear in Profiler.

Topics

- Using Annotations (p. 1934)
- Creating Annotations (p. 1935)
- Viewing Annotations in Trace Messages Profiler (p. 1936)

Using Annotations

Annotations in the Lumberyard Profiler tool flag frames in your captured data that have associated log information. By default, annotations are turned off.

To use annotations

1. To turn on annotations in the Lumberyard Profiler tool, click Configure Annotations:

![Configure Annotations dialog box]

The Configure Annotations dialog box contains a list of available annotations and their display colors. For information on creating annotations for your application, see Creating Annotations (p. 1935).

2. When you select an annotation in the dialog box, a marker and line of the same color appears in the channel display. Note that you might have to scroll horizontally to find the marker.
3. To display details for the annotations that occurred on a frame, pause your pointer on an annotation marker. In the example image, IP addresses have been redacted out.

Creating Annotations

To create an annotation, you add one or more lines of C++ logging code to your application. The added code instructs Lumberyard's logging system to include the logging information that you specify as a part of your capture. Lumberyard transforms the logged messages into annotations for you. Then, in Profiler, when you click Configure Annotations, you actually choose which system's annotations are displayed (for example, GridMate or MultiplayerProject).

To create an annotation, place a line of C++ code like the following in your application:
The first parameter is the window (that is, system) of the trace (in this case, GridMate), and the second is the content of the trace that will be shown as the annotation.

The example results in the following annotation text:

GridMate - Connection <IP_Address>|64090 => <IP_Address>|57455 (Client) (Connections=1)!

The text displays in Profiler like this:

![Profiler annotation example](image)

**Alternatives to AZ_TracePrintf**

In your code, instead of using AZ_TracePrintf, you can use AZ_Error or AZ_Warning, depending on the degree of severity that you want. AZ_TracePrintf always logs a message, but is of the lowest concern from an inspection viewpoint.

The following example uses AZ_Error:

```cpp
if (networkTableContext.ReadValue(elementIndex,forcedDataSetIndex))
{
    AZ_Error("ScriptComponent",forcedDataSetIndex >= 1 && forcedDataSetIndex <= ScriptComponentReplicaChunk::k_maxScriptableDataSets,"Trying to force Property (%) to an invalid DataSetIndex(%d).",scriptProperty->m_name.c_str(),forcedDataSetIndex);
    if (forcedDataSetIndex >= 1 && forcedDataSetIndex <= ScriptComponentReplicaChunk::k_maxScriptableDataSets)
    {
        networkedTableValue.SetForcedDataSetIndex(forcedDataSetIndex);
    }
    else
    {
        AZ_Error("ScriptComponent",false,"Trying to force Property (%) to unknown DataSetIndex. Ignoring field." , scriptProperty->m_name.c_str());
    }
}
```

In the example, if either of the error conditions occur, an annotation is created.

**Viewing Annotations in Trace Messages Profiler**

Another way to confirm that your annotations are in place is by using the Trace Messages profiler.

In the Profiler Logging channel, click the Trace messages profiler details icon to see the logging systems currently in place:
The **Trace messages** profiler instance shows all the trace messages that were generated from the start of the capture to the currently analyzed frame. Messages are shown with the oldest message at the top and the newest message at the bottom:

You can use the **Window Filter** to show the system and/or **Message Filter** to show the message text that you are interested in.

The following example, filtered by "GridMate", shows the message specified by the line of code that was added to the application:

**Using Profiler for Networking**

Profiler is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Lumberyard Profiler tool to examine how your game uses network bandwidth, including its GridMate carrier connections and replica activity. You can use network-specific profilers to drill down further into the activity of specific replica chunks, RPCs, and data sets.

**Prerequisites**

This topic assumes familiarity with Lumberyard networking and the Lumberyard Profiler tool. For information on Lumberyard networking, see Using Lumberyard Networking (p. 1990). For an introduction to the Profiler tool, see Profiler (p. 1920).

**Topics**

- Carrier Profiler (p. 1938)
- Replica Activity Profiler (p. 1939)
Carrier Profiler

The Profiler tool has a GridMate channel with Carrier and Replica activity profiler instances. You can use the Carrier profiler detail view to examine the bandwidth usage of a selected GridMate carrier connection.

To open the detail view for the Carrier profiler

- Click the Detailed Profiling Information icon for Carrier in the GridMate channel:

The Carrier profiler detail view resembles the following image:

![Carrier Profiler Detail View](image)

This view uses all of the data supplied in the capture session to show an overview of the bandwidth usage through the GridMate carrier for the selected connection. It includes the following information:

- **Total Sent/Total Received** – The total number of bytes sent and the total number of bytes received on the selected connection.
• **User Data Sent/User Data Received** – The user data sent and the user data received on the selected connection. This data does not include the overhead associated with carrier or connection maintenance.

• **Packets Sent/Packets Received** – The number of packets sent and the number of packets received.

• **Return Trip Time (Latency)** – How many seconds the packets took to make a return trip.

**Replica Activity Profiler**

You can use the Replica Activity profiler to see how much replica bandwidth your application is using.

**To open the Replica Activity profiler**

• Click the **Detailed Profiling Information** icon for **Replica activity**.

The Replica Activity profiler detail view has a pair of **Bytes Sent** and **Bytes Received** graphs at the top, a toolbar to control the display in the middle, and a table of replicas at the bottom:
This view is useful for discovering how much bandwidth a single entity is using, and for finding what information is synchronized in response to particular events for particular entities.

Two main detail views are available for replica activity: **Replica** and **Chunk Type**. The view defaults to **Replica**, but Profiler remembers your most recent choice and uses it the next time you view replica activity details.
Using Replica View

In replica view, the table shows how much data each replica used in a given frame.

To change the view to Replica

- In the toolbar, choose Replica.

Each replica is represented by its associated color in the graphs above the toolbar. Replica view includes the following information:

- **Bytes Sent** – Shows bandwidth usage in bytes sent by the object for a particular frame.

- **Bytes Received** – Shows bandwidth usage in bytes received by the object for a particular frame.

To display or hide an individual line in the graph

- Double-click the associated row in the tree.

The toolbar also offers the following options:

- **Hide All** – Hides the line graphs of all replicas in the table.

- **Show All** – Shows the line graphs for all replicas in the table.

- **Hide Selected** and **Show Selected** – Use Ctrl+click to select individual replicas in the table, and then click **Hide Selected** or **Show Selected** to hide or show the graphs for the replicas that you selected.

- **Display Range** – Determines the number of frames that are shown in the graph, with the currently selected frame in the center. You can use this option to zoom in or out on the data.

To display replica chunk details for a particular replica

- Click its details icon.

The graph shows the bytes sent and received for a replica chunk, data set, and RCP:
You can use this details view to see what replica chunk types a given replica is using, how much data each replica chunk type is using, and how much bandwidth individual data sets and RPCs are using.

**Tip**
Click **Expand All** to list all replica chunks in all replicas, and every data set and remote procedure call (RPC) in each replica chunk:
<table>
<thead>
<tr>
<th>Display Name</th>
<th>Sent Bytes</th>
<th>Received Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityReplica</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>DataSets</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>eEA_Physics</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClient1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameServerC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientDynamic</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameServerD</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientD</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameServerDynamic</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientK</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_Aspect29</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SpawnParams</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameServerA</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_Aspect30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ClientDelegatedAspects</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientG</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientB</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_Aspect31</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_Script</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientH</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameServerB</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientStatic</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cEA_GameClientP</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientO</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientI</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameClientC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eEA_GameServerStatic</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AspectProfiles</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GridMateReplicaStatus</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DataSets</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DebugName</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UpstreamSuspended</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OwnerSeq</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
To use the Replica Activity profiler tree view

- Do either of the following:
  - Select a row to highlight its corresponding line in the graph.
  - Double-click a row to display or hide the graph for the row.

The following information is available:

- **Display Name** – The debug name associated with the corresponding row of the table.

- **Sent Bytes** – The number of bytes sent for an item, including all information sent by children of the item.

- **Received Bytes** – The number of bytes received by an item, including all information received by children of the item.

**Chunk Type View**

Chunk type view shows you how much data each chunk type used in a given frame. The view is useful for seeing how much information a particular system might be using across all entities.

**To change the view to Chunk Type**

- In the toolbar on the main detail page for Replica activity, choose **Chunk Type**.

The chunk type view shows how much data a particular replica chunk type is using in a given frame:
To inspect chunk type details

- Click the details icon for the chunk type:

  The details window shows which replicas are using a chunk type's bandwidth, how much data they are using, and how much data the individual data sets and RPCs are using:
As before, you can expand the items in the tree to see detailed information about each:
Using the Profiler for CPU Usage

Profiler is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The CPU profiler gathers usage statistics about how long a function or method executed, how many times it was executed, who called it, and how much of a frame was spent on it. You can combine this information to get a systemwide view of usage, or isolate particular systems by filtering for specific threads.

To use the CPU profiler

1. To open the detail view for the CPU profiler, click the **Detailed Profiling Information** icon for the CPU profiler instance.
The CPU details view has a graph of CPU usage, a toolbar, and a tree view of calls made in a frame. Each call in the tree view has the same color as its corresponding line in the graph:

2. Pause your mouse on a line in the graph to see the call that the line represents and to display the specific value for the graph at the area near the cursor.

3. To show or hide the line graph of a row in the tree, double-click the row.

Understanding the Tree View

The CPU profiler tree view represents a call hierarchy of profiler log points (called hooks). A profiler hook that is active while another call is active shows as a child of the first hook. The hooks act as a stack: the last hook that was pushed onto the stack is the parent of the hook that was pushed onto the stack before it. The tree view has the following information:

Function

The function declaration where the profiler data point was generated.
Comment
A user-defined message that distinguishes specific events in the same function.

Excl. Time (Micro)
(Exclusive time) The time, in microseconds, spent executing this function and no other functions called by this function.

Incl. Time (Micro)
(Inclusive time) The time, in microseconds, spent executing this function and other functions called by this function.

Excl. Pct
(Exclusive percent) Exclusive time represented as a percent of total run time.

Incl. Pct
(Inclusive percent) Inclusive time represented as a percent of total run time.

Calls
The number of calls to this function.

Child Time (Micro)
The time, in microseconds, that functions that were called by this function took to execute.

Total Time (Micro)
A running total of the time, in microseconds, that was spent inside of this function.

Child Calls
How many functions this function called.

Total Calls
The running total of how many times this function was called.

Thread ID
The thread on which this function was executed

Controlling the Display
You can use the toolbar to control how the captured CPU data is displayed:

| Hide Selected | Show Selected | Hide All | Show All | Invert | Expand Tree |

Hide Selected
Hide the graph of the rows selected in tree view.

Show Selected
Show the graph of the rows selected in tree view.

Hide All
Hides the graph of all rows in the tree view.

Show All
Shows the graphs of all rows in the tree view.

Invert
Shows graphs for all rows in the tree view that are hidden; hides the graphs of all rows in the tree view that are showing.
**Expand Tree**

Expands all rows in the tree view hierarchy.

The right side of the toolbar offers more options:

- **All Threads**

  Use the thread selector to control which threads are shown in the tree view and in the graph:

  ![Thread Selector]

- **Incl. Time**

  Use this selector to choose the meaning of the time displayed.

  - **Incl. Time** – (Inclusive time) The time spent in this function inclusively.
  - **Excl. Time** – (Exclusive time) The time spent in this function exclusively.
• **Calls** – The number of times this function was called in the frame.
• **Acc. Time** – (Accumulated time) The total amount of time spent in this function up to the frame being analyzed.
• **Acc. Calls** – (Accumulated calls) – The total number of times this function was called up to the frame being analyzed.

<number> Frames

Use this selector to choose how frames of history are displayed in the graph:

![Frames selector](image)

Delta

Unused option.

Autozoom

When selected, maintains the approximate zoom level (number of frames displayed) whenever the graph changes.

Flat View

Flattens the tree of function calls (removes the hierarchical indentation), as in the following image:

![Function tree](image)

**Using Profiler for VRAM**

Profiler is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use the video memory profiler (VRAM profiler) to determine which resources are contributing most to runtime VRAM usage in your game.

The VRAM profiler records the amount of video memory used by a game, including how many memory deallocations and allocations occurred during the capture. This latter information is useful in tracking down rendering performance bottlenecks.

You can also use the memory usage information from VRAM profiler to determine your game's minimum PC GPU (graphics processing unit) memory requirements, or to determine whether your game will run out of memory on a console or mobile device.

**Topics**
- Notes (p. 1952)
- Understanding the Captured Data (p. 1952)
- Inspecting the Data (p. 1953)

**Notes**

The VRAM profiler has the following attributes:

- The VRAM profiler has no graph view or tree view.
- The only supported export format is .csv. For steps on saving Profiler data to a .csv file, see Exporting Data (p. 1933).
- Lumberyard uses a variety of memory pooling schemes, so the actual allocated amount of VRAM is slightly more than what is reported.

**Understanding the Captured Data**

The following image shows how your saved .csv file appears in a spreadsheet application:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Category</td>
<td>Number of Allocations</td>
<td>Memory Usage</td>
</tr>
<tr>
<td>2</td>
<td>Texture</td>
<td>449</td>
<td>440542078</td>
</tr>
<tr>
<td>3</td>
<td>Buffer</td>
<td>1034</td>
<td>14778088</td>
</tr>
<tr>
<td>5</td>
<td>Resource Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$RT_ShadowPool</td>
<td>67108864</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CachedShadowMap_0</td>
<td>35515592</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$AutoDownload_4</td>
<td>16777216</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$AutoDownload_3</td>
<td>16777216</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$AutoDownload_5</td>
<td>16777216</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>HeightMapAO_Depth_1</td>
<td>11184800</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>HeightMapAO_Depth_0</td>
<td>8388608</td>
<td></td>
</tr>
</tbody>
</table>

The captured data contains essentially two tables of information: an overview of memory allocation and usage (divided between texture and buffer assets), and a list of resources with the amount of VRAM that was allocated for each during the capture.

Detailed information about each heading follows.

**Category**

Indicates the type of allocation:
- **Texture** – Includes texture assets, dynamically generated textures, and frame buffers.
- **Buffer** – Includes vertex and index buffers, constant buffers, and other runtime buffers.
Number of Allocations

The number of allocation events recorded. When the capture starts, all active allocations are sent to the profiler as a starting number. Any new allocations or deallocations will increase or decrease this number.

Memory Usage

The total size, in bytes, of VRAM used.

Resource Name

The name and full path of the allocated resource. A resource name without a path usually denotes a runtime engine resource.

VRAM Allocation Size

The size, in bytes, of the allocation.

Inspecting the Data

When you first open the spreadsheet, the data is unordered. To sort the data, you can use a spreadsheet application:

To quickly and easily identify the largest offending assets or runtime resources, sort by VRAM Allocation Size in descending order, or by Resource Name from A to Z:
Negative VRAM Allocation Sizes

Some fields may have a negative number for **VRAM Allocation Size**, as in the following image:

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>VRAM Allocation Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConstantBuffer</td>
<td>-256</td>
</tr>
</tbody>
</table>

These important occurrences show that a VRAM deallocation event occurred during the capture. If you observe a large number of deallocation entries over a short time period, your game might be experiencing a significance decrease in performance. To improve your game's performance across all operating systems, you should aim to have as few idle per-frame VRAM allocations and deallocations as possible.

**Why Some Textures Are Not Reported in the .csv File**

If you see a lot of allocations named `StreamingTexturePool` or entries like `$TexturePool_9_0000000002C59248`, this means the texture streaming system is active. The texture streaming system allocates all textures by default into a variety of cached texture pools. The VRAM profiler reports the size of the active streaming pools and not the names of the actual texture assets. To obtain the names and sizes of the allocated and loaded textures, set `r_TexturesStreaming=0` in your system configuration file, and then do another capture. This setting disables the texture streaming system and causes the true sizes of the texture allocations to be reported.

**Note**

In this situation, it is advisable to do two captures: one with `r_TexturesStreaming` enabled, and one with it disabled. When texture streaming is enabled, your VRAM usage is less because of texture eviction and the loading of lower resolution mipmap levels. The memory reporting is more accurate when texture streaming is enabled, but you get a much clearer view of your worst-case memory usage when texture streaming is disabled.

**Using GridHub**

GridHub is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
diagnostic and debugging tools Profiler.exe or LuaIDE.exe (located in the \dev\Bin64 directory), GridHub launches as a background process in Windows and enables their functionality. For more information about Profiler, see Profiler (p. 1920).

**Note**
Because GridHub listens for connections on the loopback address (127.0.0.1), you must run GridHub on the same computer as the target application.

**Topics**
- Registering an Application in GridHub (p. 1955)
- Viewing and Configuring GridHub (p. 1955)
- Troubleshooting GridHub (p. 1958)

**Registering an Application in GridHub**

To register an application in GridHub so that Profiler can capture information from the application, add AzFramework::TargetManagementComponent to the application's SystemComponent.

**Note**
Lumberyard's built-in applications already have this component added by default.

**Viewing and Configuring GridHub**

When you launch Profiler.exe or LuaIDE.exe, GridHub starts automatically and is represented by a globe icon in the Windows taskbar.

**To view and configure GridHub**

1. In the Windows taskbar, right-click the globe icon and choose **Show**:

![Show Icon](image)

The GridHub window has a configuration bar, a connections pane, and pane for viewing log messages:
2. You can use the configuration toolbar to view or change GridHub configuration:

The toolbar options are as follows:

**Session port** – Specifies the port on which GridHub listens for discovery requests.

**Connection slots** – Specifies the maximum number of applications that can be connected concurrently to GridHub.

**Hub name** – The name of your hub. By default, this is the name of the local computer.

**Note**

The name of the hub must be the neighborhood name to which the TargetManagementComponent connects.

**Enable Disconnection Detection** – Specifies whether the connection to GridHub is terminated when the source fails to respond.

**Add to Windows startup folder** – Specifies whether GridHub starts automatically when Windows starts.
Log activity – Starts or stops logging.

Start/Stop – Starts or stops GridHub. When GridHub is off, no connections are discovered or maintained.

3. When GridHub and your target application are active, your target application appears in the GridHub Connections list:

The columns in the Connections list provide the following information:

ID – The identifier of the connected application.

Name – The name of the connected application.

Connection ID – The identifier of the connection between GridHub and the application.

IsHost – Whether or not the connection is the connection host

IsLocal – Whether or not the connection is local.

IsReady – Whether or not the application is ready to handle further connections.

4. Use the Output window to see the log messages that GridHub generates as it manages connections:

When GridHub is terminated, the connections it established are also terminated.
Troubleshooting GridHub

If you experience difficulty using GridHub, check the following:

- Make sure that the neighborhood name in TargetManagerComponent is the same as the one in GridHub.
- Make sure that the port that GridHub is listening on is the same port as the one specified for TargetManagementComponent.
- Make sure that all applications are running on the same computer. The GridHub socket is bound to the loopback address 127.0.0.1.

Using AZ Test Scanner

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The AZ test scanner is a tool for running unit tests that are built into Lumberyard libraries and executables. This tool simplifies testing by automatically finding libraries and executables to test. It also provides flexibility for programmers to focus on testing the parts of Lumberyard that they care about.

The AZ test scanner has two components:

- An AZ test runner executable that loads libraries to test and capture the test results
- An AZ test Python module that performs the scanning and reporting functions

Creating Unit and Integration Test Builds

Unit and integration tests are not included in Lumberyard builds by default as they increase the overall size of a game project. Test code can also have unexpected effects on performance. To build components with tests included, you can use a special test variant that works with each configuration.

To create test builds, use the Waf build system (p. 63) in the same way that you create regular builds. The only difference is that you add test to the platform. You can create a test build on Windows using one of the following examples:

```bash
// Build with tests using debug configuration. Outputs to the Bin64vc141.Debug.Test folder.
lmbr_waf.bat build_win_x64_vs2017_debug_test -p all

// Build with tests using profile configuration. Outputs to the Bin64vc141.Test folder.
lmbr_waf.bat build_win_x64_vs2017_profile_test -p all

// Build with tests using debug configuration. Outputs to the Bin64vc142.Debug.Test folder.
lmbr_waf.bat build_win_x64_vs2019_debug_test -p all

// Build with tests using profile configuration. Outputs to the Bin64vc142.Test folder.
lmbr_waf.bat build_win_x64_vs2019_profile_test -p all
```

Note

Only Windows debug and profile builds are supported for testing. Other platforms are not supported. Release builds are not supported either.
Running Unit and Integration Test Builds

A completed test build includes the file `AzTestRunner.exe` in the `\Bin64\vc141\Test` folder. Although you can use this to run tests, we recommend that you use the test scanner that uses `AzTestRunner.exe` in an automated manner.

You have two ways to use the scanner:

- Include the AZ test module in your Python path: `python -m aztest`.
- Use the `lmbr_test.cmd` script located in the Lumberyard `\dev` folder. This automatically includes the AZ test module in your Python path and sends all script parameters to the module.

The following example uses the `lmbr_test.cmd` scripts. The scanner has several options but only requires one parameter to operate: the build directory to scan. You can use the following command to scan your entire test build:

```
// Scan entire test build and run all found tests
lmbr_test.cmd scan --dir Bin64\vc141\Debug\Test
```

**Note**
The default scan tests libraries only. It does not attempt to test any executables it finds. This is because executables that are not set up to run tests interrupt the scanner until you close the application.

The scanner produces three types of files. All files are created in the current working directory from which the scanner is called:

- The `aztest.log` file that contains a log of all test output
- Several `.xml` files that contain the test results of each library and executable that has tests, time stamped by default
- An `.html` file that contains a summary of the test results from the entire scan, time stamped by default

The full list of options is shown as follows:

The scanner runs only unit tests by default. This is because unit tests are designed to be fast and do not rely on engine resources. To run integration tests instead, use the `--integ` flag when calling the scanner:

```
// Scan test build and run integration tests on CrySystem.dll
lmbr_test.cmd scan --dir Bin64\vc141\Debug\Test --only CrySystem.dll --integ
```

**Note**
For best results, run integration tests on a single library or use an allow list. Scanning the full build might take hours to complete.

<table>
<thead>
<tr>
<th>Option</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--dir, -d</td>
<td>Yes</td>
<td>The directory to scan for tests.</td>
</tr>
<tr>
<td>--runner-path</td>
<td>No</td>
<td>Path to the AZ test runner executable (the default is to look in the directory specified by --dir).</td>
</tr>
</tbody>
</table>
Using AZ Test Scanner

<table>
<thead>
<tr>
<th>Option</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--add-path</td>
<td>No</td>
<td>Adds path to system path before running tests; used for resolving library or executable dependencies.</td>
</tr>
<tr>
<td>--output-path</td>
<td>No</td>
<td>Sets the path for output folder prefix (the default is <code>\dev\TestResults</code>).</td>
</tr>
<tr>
<td>--integ, -i</td>
<td>No</td>
<td>If set, runs integration tests instead of unit tests.</td>
</tr>
<tr>
<td>--no-timestamp</td>
<td>No</td>
<td>If set, removes the time stamp from output files.</td>
</tr>
<tr>
<td>--wait-for-debugger</td>
<td>No</td>
<td>If set, tells the AZ test runner executable to wait for a debugger to be attached before running tests.</td>
</tr>
<tr>
<td>--bootstrap-config</td>
<td>No</td>
<td>Path to a JSON configuration file for bootstrapping applications required by libraries.</td>
</tr>
<tr>
<td>--limit, -n</td>
<td>No</td>
<td>Sets a limit for the maximum number of modules to scan.</td>
</tr>
<tr>
<td>--only, -o</td>
<td>No</td>
<td>Sets a filter to run tests on only the specified library or executable name.</td>
</tr>
<tr>
<td>--whitelist-file</td>
<td>No</td>
<td>Path to a new line-delimited file used as an inclusion list. The new line-delimited file allows for regular expressions when matching.</td>
</tr>
<tr>
<td>--blacklist-file</td>
<td>No</td>
<td>Path to a new line-delimited file used as an exclusion list. The exclusion list takes precedence over the inclusion list. The new line-delimited file allows for regular expressions when matching.</td>
</tr>
<tr>
<td>--exe</td>
<td>No</td>
<td>If set, causes the scanner to call executables for testing. (The default is to test only libraries.)</td>
</tr>
</tbody>
</table>

The scanner also accepts additional parameters that are passed to the testing framework. For Lumberyard, GoogleTest, and GoogleMock for C++ are used for unit testing. You can enter parameters in the scanner command line as shown in the following example:

```
// Scan CrySystem.dll and shuffle the test order before running
lmbr_test.cmd scan --dir Bin64vc141.Test --only CrySystem.dll --gtest_shuffle
```

The scanner can also be called as a chained command using Waf. This means that you can build tests and run them using a single command line. The Waf command `run_tests` calls the scanner on the most recent build folder. For example:

```
// Build a debug test build and then run tests in it
lmbr_waf.bat build_win_x64_vs2017_debug_test -p all run_tests
```

The `run_tests` command automatically points to the `\Bin64vc141.Debug.Test` folder to scan. It also uses the `all` option for inclusion. The build step does not require the use of `run_tests`; it always matches the last build. You can also send all of the scanner parameters through using `--test-params:

```
// Run tests on the last build with additional parameters (use quotes to capture as string)
```

Version 1.28
1960
Writing Tests for AzTest

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Automated testing is important for any game project. To perform automated testing in Lumberyard, you can use AzTest and the AZ test scanner. This document shows you how to write the tests to build and run. For information on building and running unit and integration tests, see Using AZ Test Scanner (p. 1958).

Lumberyard and AzTest use the GoogleTest and GoogleMock frameworks for unit and integration tests. When you write your tests, we recommend that you consult the corresponding documentation.
Testing code with AzTest has three steps: Configure your module for testing, write the tests, and define global environments.

**Configuring Your Module for Testing**

If you want to add tests to a module or a gem that is included with Lumberyard, the configuration has already been done for you and you can skip this step. If you want to add tests to non-Lumberyard modules, projects, or tools, then you must configure them to use AzTest.

All modules and libraries in Lumberyard must be configured for use with Waf. If you are unfamiliar with configuring build modules in Waf, see Adding a Build Module (p. 125) before continuing.

**To configure your module for Waf**

1. Create a `.waf_files` content file that lists the files for the test. A separate `.waf_files` content file for test files keeps the test files from being compiled in normal builds. Your file name should make it clear that it contains only test files. Existing Lumberyard modules and gems use the naming convention `<module_name>_test.waf_files`.

   The following example shows a `.waf_files` file for a HelloWorld module.

   ```
   {
     "none":
     {
       "Tests":
       [  
        "HelloWorldTestMain.cpp"
       ]
     }
   }
   ```

   After you create the file, you must reference it in your module's `wscript`. To specify that the file is intended only for test builds, you must add `test_all_file_list` to the `wscript` configuration, as in the following example.

   ```
   # HelloWorld wscript
   def build(bld):
     bld.CryEngineModule(
       target    = 'HelloWorld',
       vs_filter = 'Engine',
       file_list = 'helloworld.waf_files',
       use       = ['AzCore'],

       # Testing
       test_all_file_list = 'helloworld_test.waf_files'
     )
   ```

2. Create a test hook for AZ Test Scanner. How you create a test hook depends on whether you want to build a dynamic library or an executable file. Static libraries are not currently supported for testing.

   - **To create a test hook for a dynamic library**

     Because the tests built into a dynamic library are not exposed, you must expose a separate test function for the AZ test scanner. To expose the test function, use a convenience macro that AzTest provides in a test-only * _test.waf_files file, as in the following example.

     ```
     // HelloWorldTestMain.cpp
     #include <AzTest/AzTest.h>
     ```
AZ_UNIT_TEST_HOOK(); // Runs unit tests
AZ_INTEG_TEST_HOOK(); // Runs integration tests

For differences between unit tests and integration tests, see Creating Unit Tests and Integration Tests (p. 1964).

- **To create a test hook for an executable file**

To build an executable file, you must expose a test function and modify the main function of the executable to run tests instead of normal program functions. The test function informs the AZ test scanner that tests have been included in the executable and that it is safe to continue.

The following example shows how to modify your main function:

```cpp
// Main.cpp
#if defined(AZ_TESTS_ENABLED)
#include <AzTest/AzTest.h>
DECLARE_AZ_UNIT_TEST_MAIN()
#endif

int main(int argc, char* argv[])
{
#if defined(AZ_TESTS_ENABLED)
INVOKE_AZ_UNIT_TEST_MAIN();
#endif
// Rest of your program
}
```

**Note**
Because the AZ_TESTS_ENABLED definition is defined only in test builds, it is a convenient definition to use in test-only code.

3. If necessary, use the `use` parameter in the `wscript` file to link AzTest to the module.

**Note**
In most cases, you do not have to perform this step. Waf automatically links AzTest in test builds for almost all build modules, including gems. The LumberyardApp build module is not linked automatically. If you build a Lumberyard app, you must link it manually.

As with the `*_test.waf_files` files, you must use AzTest only in a test build. To specify this when linking, add `test_all_use` to the `wscript` configuration, as in the following example.

```wscript
# HelloWorld wscript
def build(bld):
    bld.CryEngineModule(
        target    = 'HelloWorld',
        vs_filter = 'Engine',
        file_list = 'helloworld.waf_files',
        use       = ['AzCore'],

        # Testing
        test_all_file_list = 'helloworld_test.waf_files',
        test_all_use       = ['AzTest'],
    )
```

After you have performed these three steps, you can create a test build and test it with the AZ test scanner. Your HelloWorld library or executable should appear in the report, although at this point it does not have tests.
Creating Tests

After your module is configured for testing and visible to the AZ test scanner, you can start writing tests for it. Detailed information about writing tests is available in the GoogleTest and GoogleMock documentation. The following information describes characteristics specific to AzTest.

Creating a Simple Example Test

When using AzTest to write tests, include only the `AzTest.h` file in your test files instead of GoogleTest and GoogleMock, as in the following example.

```
// HelloWorldTests.cpp
#include <AzTest/AzTest.h>
#include "HelloWorld.h"

TEST(HelloWorldTests, HelloWorld_ReturnsHelloWorld)
{
    // Call HelloWorld() and compare it to the expected C-string
    ASSERT_STREQ("Hello World!", HelloWorld());
}
```

Creating Unit Tests and Integration Tests

Unit tests are designed to be run in isolation. Unit tests are expected to pass without requiring Lumberyard's engine systems or other assets. Integration tests are designed to be run with the game engine and have access to the engine's systems or assets.

To prevent unexpected failures, the different kinds of tests must be kept separate. Lumberyard does this by providing different hooks for each kind of test. Declare the tests as unit or integration, and call the AZ test scanner accordingly.

For tests that do not require fixtures, simply replace the `TEST` macro with `INTEG_TEST`, as in the following example.

```
// HelloWorldTests.cpp
#include <AzTest/AzTest.h>
#include "HelloWorld.h"

INTEG_TEST(HelloWorldTests, MessageOfTheDay_ReturnsCurrentMessageOfTheDay)
{
    // Call MessageOfTheDay() and compare it to the expected C-string
    // This is an integration test because MessageOfTheDay is stored in a database
    MessageDatabase messageDatabase = HelloWorld::GetTestMessageDatabase();
    ASSERT_STREQ("This is your message for the day!", messageDatabase.MessageOfTheDay());
}
```

For tests that do use fixtures, add the `Integ_` prefix to the fixture name to indicate that it is to be used only for integration tests, as in the following example.

```
// HelloWorldTests.cpp

class Integ_MessageDatabaseTests : public ::testing::Test
{
    protected:
        void SetUp() override
        {

```
m_messageDatabase = HelloWorld::GetTestMessageDatabase();
}
MessageDatabase m_messageDatabase;

TEST_F(Integ_MessageDatabaseTests, MessageOfTheDay_ReturnsCurrentMessageOfTheDay)
{
    ASSERT_STR("This is your message for the day!", messageDatabase.MessageOfTheDay());
}

**Note**

Fixtures cannot be shared between unit and integration tests. If you want to use the same fixture for both kinds of tests, create a base fixture and subclass for each type of test.

### Using Global Environments

It is often useful to create some global variables or mocks for testing the module. In Lumberyard, this might mean creating memory allocators or a test system environment. Because AzTest extends the support that GoogleTest provides, you must use the AzTest environments.

The following example shows how to define an AzTest global environment:

```cpp
// HelloWorldTestMain.cpp
#include <AzTest/AzTest.h>

class HelloWorldEnvironment : public AZ::Test::ITestEnvironment
{
protected:
    void SetupEnvironment() override
    {
        // Environment setup here
    }
    void TeardownEnvironment() override
    {
        // Environment teardown here
    }
}
```

To initialize the environments, use the test hooks that were mentioned previously. Because the macros are variadic, you can initialize as many environments as you require. The following example initializes the environment in the previous example:

```cpp
// HelloWorldTestMain.cpp
#include <AzTest/AzTest.h>

class HelloWorldEnvironment : public AZ::Test::ITestEnvironment
{
protected:
    void SetupEnvironment() override
    {
        // Environment setup here
    }
    void TeardownEnvironment() override
    {
        // Environment teardown here
    }
}
AZ_UNIT_TEST_HOOK(new HelloWorldEnvironment);

**Note**
You must dynamically allocate environments before you use the macro. This gives you full control over how environments are created at runtime. The order of environments is also significant. Environments earlier in the list are initialized earlier and removed later. This is important if a global environment depends on another environment that already exists.

## Debugging Issues

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard provides the following built-in debugging and profiling tools that you can use to locate and fix performance issues.

- **Character skeleton debugging** – Use the `p_draw_helpers` console variable to debug character skeleton issues.
- **Cinematics debugging** (p. 1632) – Debug cinematics issues.
- **Mannequin debugging** – Debug Mannequin system issues.
- **Particle debugging** (p. 1578) – Debug particles.
- **Vegetation debugging** (p. 1313) – Debug vegetation objects.

### Topics

- Crash logging (p. 1966)
- Using Console Debug Views (p. 1967)
- Using the `sys_asserts` Console Variable (CVAR) (p. 1968)

### Crash logging

Logging and reporting for crashes that occur in the Lumberyard Editor or your Lumberyard game can be turned on by modifying the `sys_dump_type` console variable. The type of crash log information generated depends on the value:

- **0** – Disable crash reporting.
- **1** – Generate a stack trace on crash
- **2** – Generate a stack trace and limited variable information.
- **3** – Full crash dump of the stack trace, variable information, and all memory.

Crash logs are written to the `Cache\project_name\platform\user\log\error.log` file.

By default, only the call stack for the main thread is in a crash dump. In order to get the stack trace of auxiliary threads, set the `sys_dump_aux_threads` console variable to 1.
Using Console Debug Views

The viewport window displays debugging information by default when you are in gameplay mode (Ctrl +G). You can toggle this information on or off by pressing the tilde (~) key. Use the following console variables and values to generate viewing modes in the viewport that are useful for debugging.

- **e_Camerafreeze 1** – Freezes the camera to see what is rendered from the camera's point of view and what is occluded. Also useful for debugging object culling and LOD.
- **e_DefaultMaterial 1** – Applies a uniform, flat, gray material to every surface in the level.
- **e_memoryProfiling 1** – Prints onscreen statistics for the amount of GPU memory that textures and buffers use.
- **e_TerrainBBoxes** – Displays terrain bounding boxes.
- **p_debug_joints 1** – Shows the mass of objects in kilograms and the joint that is linked to the object. To display joints, you must first enable **p_draw_helpers 1**.
- **p_draw_helpers 1** – Shows physics proxy meshes in addition to the render geometry.
- **r_DisplayInfo 0** – Disables the debugging text.
- **r_DisplayInfo 1 | 2 | 3 | 4** – Enables various levels of debugging text. Displays memory consumption, frame rate, triangle count, visible light sources, and drawcall count. Use a value of 2 to display more detailed information. Use a value of 3 to display only frames per second (FPS) and frame time in milliseconds.
- **r_wireframe 1 | 2** – Use a value of 1 to draw the level in wireframe mode. Use a value of 2 to draw the level in vertex mode. Both include the objects that are hidden from view.
- **r_ShowLines 2** – Overlays the wireframe only on the front-facing geometry. Anything behind this geometry isn't rendered.
- **r_TexBindMode 6** – Applies a uniform, flat, gray material with normal map information to every surface in the level.

### VRAM Usage: 1409.4MB

- Texture / Rendertargets - Size: 744.7MB - Allocations: 157
- Texture / Texture Assets - Size: 595.8MB - Allocations: 270
- Texture / Dynamic Textures - Size: 0.1MB - Allocations: 2
- Buffer / Vertex Buffers - Size: 12.0MB - Allocations: 189
- Buffer / Index Buffers - Size: 32.5MB - Allocations: 70
- Buffer / Constant Buffers - Size: 0.2MB - Allocations: 212
- Buffer / Other Buffers - Size: 24.1MB - Allocations: 8

Using DebugDraw Console Variables

Use the following console variables and values to display information about your level.

- **e_DebugDrawLodMinTriangles 1000** – Do not draw debug text for objects with less than 1000 triangles.
- **e_DebugDraw 1** – Displays the name of the .cfg used, polycount, and LOD.
- **e_DebugDraw 2** – Displays a color-coded polygon count.
- **e_DebugDraw 3** – Displays a color-coded LOD count. Flashing color indicates a single LOD.
- **e_DebugDraw 4** – Displays object texture memory usage.
- **e_DebugDraw 5** – Displays a color-coded number of render materials.
• e_DebugDraw 6 – Displays ambient color.
• e_DebugDraw 7 – Display triangle count, number of render materials, and texture memory.
• e_DebugDraw 8 – Displays RenderWorld statistics (with view cones).
• e_DebugDraw 9 – Displays RenderWorld statistics (with view cones without lights).
• e_DebugDraw 10 – Displays render geometry with simple lines and triangles.
• e_DebugDraw 11 – Displays render occlusion geometry.
• e_DebugDraw 12 – Displays render occlusion geometry without render geometry.
• e_DebugDraw 13 – Displays occlusion amount (used during AO computations).
• e_DebugDraw 15 – Displays helpers.
• e_DebugDraw 16 – Displays debug gun.
• e_DebugDraw 17 – Displays streaming information (buffer sizes).
• e_DebugDraw 18 – Displays streaming information (required streaming speed).
• e_DebugDraw 19 – Displays physics proxy triangle count.
• e_DebugDraw 20 – Displays object instant texture memory usage.
• e_DebugDraw 21 – Displays animated object distance to camera.
• e_DebugDraw 22 – Display object's current LOD vertex count.
• e_DebugDraw 23 – Display object in red if it is casting a shadow.
• e_DebugDraw 24 – Display objects with 0 LOD with red text.
• e_DebugDraw 25 – Display objects with 0 LOD with red text and objects with 1 LOD with blue text.

Using GBuffer Console Variables

Use the following console variables and values to display materials, colors, shadows, albedo, and other characteristics in your level.

• r_DebugGBuffer 1 – Shows normals of all assets in the level.
• r_DebugGBuffer 2 – Shows the roughness or glossiness of the surfaces.
• r_DebugGBuffer 3 – Shows the specular color of materials.
• r_DebugGBuffer 4 – Shows the albedo of all surfaces in the level.
• r_DebugGBuffer 5 – Shows the lighting model in the level. Gray = standard | Yellow = transmittance | Blue = POM self-shadowing.
• r_DebugGBuffer 6 – Shows the translucency values that are set on assets in the level. Black = none.
• r_DebugGBuffer 7 – Shows self-shadowing of materials that use Offset Bump mapping or Parallax Occlusion Mapping.
• r_DebugGBuffer 8 – Shows in red and yellow any asset that uses SSS. The brighter the color, the higher the SSS index.
• r_DebugGBuffer 9 – Shows whether specular colors are in a reasonable range as follows:
  • Blue – The specular color is too low.
  • Orange – The specular color is too high for dielectric materials.
  • Pink – The specular color is valid only for rusted or oxidized metals.

Using the sys_asserts Console Variable (CVAR)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Manage assert notifications in Lumberyard with the `sys_asserts` console variable. The following table shows the possible values and their meanings.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore assert condition checks. Assert expressions are not evaluated. This option offers the best performance of all of these values.</td>
</tr>
<tr>
<td>1</td>
<td>If asserts and assert call stacks are available, they are logged and printed to the console or terminal. This is the default value.</td>
</tr>
<tr>
<td>2</td>
<td>If asserts and assert call stacks are available, they are logged and printed to the console or terminal. This value displays a dialog box with options to ignore the current assert, ignore all asserts, or break on the assert.</td>
</tr>
</tbody>
</table>

**Note**
Prior to Lumberyard version 1.21, when a debugger was attached and an assert was triggered, the debugger triggered a breakpoint on the assert. In the new implementation, asserts print to the log at setting `sys_asserts=1` and display a dialog box at setting `sys_asserts=2`. This gives you the option to break on the assert rather breaking automatically.

**Example Outputs**

`sys_asserts=1` produces output similar to the following:

```
(System) - Trace::Assert
e:\dev\Code\CryEngine\Cry3DEngine\Particle.cpp(1289): (68792) 'void __cdecl CParticle::Update(const struct SParticleUpdateContext &,float,bool)'
(System) - e:\dev\code\cryengine\cry3dengine\particle.cpp (1290) : CParticle::Update
(System) - e:\dev\code\cryengine\cry3dengine\particlecontainer.cpp (777) : CParticleContainer::UpdateParticleStates
(System) - e:\dev\code\cryengine\cry3dengine\particlecontainer.cpp (731) : CParticleContainer::UpdateParticles
(System) - e:\dev\code\cryengine\cry3dengine\particleemitter.cpp (87) : <lambda_11fc931574fd38d67807576e751a0e04>::operator()
```

`sys_asserts=2` opens a dialog box like the following:
Debugging Issues

The following table describes the options for the **Assert** dialog box.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore</td>
<td>Ignores the current assert and continues running the application. The same assert no longer triggers the dialog box to display.</td>
</tr>
<tr>
<td>Ignore All</td>
<td>Prevents the current assert and all future asserts from displaying a dialog box. To prevent decreases in performance, prints debug information to the log only after completion.</td>
</tr>
<tr>
<td>Break</td>
<td>Breaks on the assert. If a debugger is attached, creates a breakpoint and breaks at the breakpoint in the debugger. If a debugger is not attached, stops the application.</td>
</tr>
</tbody>
</table>

**Setting the Assert Level at Initialization**

To set the assert level at engine initialization, add an entry to a project's `game.cfg` file. The following example shows a `game.cfg` file for the SamplesProject.

```plaintext
sys_game_name = "SamplesProject"
sysLocalization_folder = Localization
cs_useIMG_CAF = 0
collision_classes = "Ship=0,Shield=1,Asteroid=2"
r_DisplayInfo=3
sys_asserts=2
```

**Setting the Assert Level at Runtime**

You can set the `sys_asserts` console variable at runtime in the console window. The following image shows an example.
Setting the Assert Level for Mobile Devices

When debugging mobile platforms, you can use the Windows-based Universal Remote Console (p. 3209) to set the assert level in the command-line window of the application.
Setting an Assert in Source Code

To add an assert in source code, use the `AZ_Assert` macro.

```c
AZ_Assert(m_useCount >= 0, "AssetData has been deleted")
```

For more information, see Tracing (p. 31).

Tracking File Access

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

It's possible to track invalid file reads that occur during game run time. The error message Invalid File Access occurs when an attempt is made to read or open files from a thread that is not the streaming thread. These file access operations can cause stalls that can be quite severe.

**Note**

Only access attempts from the main thread and render thread are logged. This feature is disabled in RELEASE builds.

CVars

The following console variables enable different options for tracking file access.

```caps
sys_PakLogInvalidFileAccess
```

Version 1.28

1972
1 (default):

- Access is logged to game.log.
- Generates a perfHUD warning.
- The warning is displayed in red in the upper left corner of the screen.
- A 3 second-stall in non-release builds is induced.

**sys_PakMessageInvalidFileAccess**

- When a file is accessed, creates a popup dialog on the PC. At this point, you can choose to break into the debugger, or continue.

### Where invalid access is defined

The points which define when a file access attempt is considered invalid are set by implementing ICryPak::DisableRuntimeFileAccess to return true or false. The points may need to be tweaked for single player and multiplayer games.

### Exceptions

To add exceptions to file access tracking so that you can ignore files like game.log, create an instance of CDebugAllowFileAccess in the scope which accesses the file.

### Resolving file access callstacks

The files that you collect with pak_LogInvalidFileAccess must have their callstacks resolved. To do this requires the following tools from the XenonStackParse folder of the Tools directory:

- The .pdb files from the build
- The XenonStackParse tool
- The ProcessFileAccess.py helper script

The directory structure for running ProcessFileAccess.py should resemble the following:

```
<Root>
    --> XenonStackParse
    --> FileAccessLogs (this folder should contain the .pdb files)
    -------> Processed (this folder contains the output from XenonStackParse)
```

Run ProcessFileAccess.py from the FileAccessLogs directory (XenonStackParse uses the working directory to search for the .pdb files). The script creates a folder called Processed and a file within it that contains the resolved callstack for each of the log files.

---

**Lumberyard UI 2.0 Developer Documentation**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Introduced with Amazon Lumberyard version 1.25, UI 2.0 is the user interface framework used by Lumberyard's editors and tools. It uses the C++ Qt UI library and can be extended to support your own customizations and modifications.

---

Version 1.28
1973
What is Lumberyard UI 2.0?

UI 2.0 is designed to enable easy-to-use workflows and help your game designers and developers translate their ideas into quick simulations. When developing the new Lumberyard UI, we focused on these four tenets:

- Efficiency
- Accessibility
- Familiarity
- Modularity

To achieve this, we provided a complete set of Qt-based user interface building blocks so that you can create tools to fit your own needs, while still employing a coherent and standardized UI experience throughout the Lumberyard Editor.

When creating tools, developers have access to the controls that we're now using throughout Lumberyard, which includes the following elements:

- Breadcrumbs
- Browse edit (*input field with button*)
- Button
- Card
- Check box / toggle switch
- Color picker and color label
- Combo box
- Filtered search widget
- Gradient slider
- Hyperlinks
- Line edit
- Menu
- Progress indicators
- Radio buttons
- Reflected property editor
- Scrollbar
- Segment control
- Slider
- Sliders with combo box
- Spin-box (*Line edit with drag support*)
- Vector input spin-box
- Splitter
- Styled dock widget
- Tabs
- Table view
- Toolbar
- Tree view
- Typography
UI 2.0 overview

With Lumberyard 1.25 and later versions, developers can use Lumberyard's custom Qt widget library to create their own developer tools and modify existing tools to blend into the user experience of the rest of the Lumberyard Editor. We have provided a set of styles and user interaction patterns that are applied on top of the Qt framework, which is the C++ based library Lumberyard relies on for its UI. For more information about how UI 2.0 works and UI component development, see the Amazon Lumberyard UI Extensions Guide

Review the UI 2.0 components

Before you begin, make sure that you have Lumberyard 1.25 or later installed on your machine.

To review the UI components

1. Open File Explorer and go to your build directory. (For example, C:\{your-lumberyard-installation-path}\{version}\dev\Bin64vc141\ or Bin64vc142\.)
2. Double-click on **AmazonQtControlGallery.exe** to open the **UI 2.0 Qt Control Gallery** app.
3. Use the main dropdown menu to navigate through the pages. For each component, the right column shows a live example of how the widget will look and work inside the Lumberyard Editor or for your tool. The left column displays the code used to produce that result.

**Note**
The Gallery does not cover how to set up a window for your tool, or the basics of how the Qt library works. Its primary objective is to illustrate the attributes of the Lumberyard Component Gallery.
Object serialization

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Lumberyard engine offers object serialization to persist objects between sessions, transmit them between clients, or work with objects between the editor and the engine. The JSON-based serialization system is designed to be human-readable and -editable, while the XML system is used by many existing Lumberyard tools. Lumberyard also offers binary serialization, used internally by the Asset Processor.

In this section of the documentation, you'll learn how to register classes and enums for serialization, and how to work with JSON serializers. For more information on XML serialization for use with Lumberyard tools, see Serialization Context (p. 1033) and Reflecting a Component for Serialization and Editing (p. 977).

Register objects for serialization

Serialization in Lumberyard is done by registering classes with a serialization context, which takes information about the provided class and uses reflection mechanisms to determine which class members to emit and their types. Serialization is controlled through the AZ::SerializeContext class, declared in AZCore/Serialization/SerializeContext.h as part of the AzCore library.

Serialization requires access to an AZ::ReflectContext instance that can be safely cast to a AZ::SerializeContext object through the AzCore reflection system. There's a globally managed serialization context within the Lumberyard engine that you can retrieve through the AZComponentApplicationBus.

```
AZ::SerializeContext* serializeContext = nullptr;
AZ::ComponentApplicationBus::BroadcastResult(serializeContext,
&AZ::ComponentApplicationBus::Events::GetSerializeContext);
```

**Warning**
When using the global serialization context, only register an object for serialization in a Reflect function call. Registering outside of this function can cause race conditions. If you need to register for serialization at any other time, use a custom serialization context.

Register classes

Classes are registered on a serialization context with the AZ::SerializeContext::Class<T>() method, using the type T to determine which class to register. In order to be serialized, the class must be a specialization of AzTypeInfo registered with the AZ_TYPE_INFO_SPECIALIZE() macro, or have RTTI information set with the AZ_RTTI macro. The AZ::SerializeContext::Class<T>() method returns an AZ::SerializeContext::ClassBuilder object, which is used to store version and field information for the class.

```
AZ::SerializeContext::ClassBuilder

Version(unsigned int version, VersionConverter converter = nullptr)
```

Sets version information for the serialization.
Register objects

- **version** – The version of the class. Whenever the internal structure of the class changes, the version should be updated.
- **converter** – Converter function which translates older versions of the class to the provided version.

```cpp
template<class ClassType, class FieldType> Field(const char* name, FieldType ClassType::* address, AZStd::initializer_list<AttributePair> attributeIds = {})
```

Tags a field in a class for storage.
- **name** - The name to store the field as. Field names for the same class must be unique. Matching the member name isn't required.
- **address** - The address of the field to store, as a pointer to member or offset from the start of an instance of `ClassType`. If a pointer to member is used, all type information is inferred.
- **attributeIds** - Associate other attribute objects with this field.

```cpp
template<class ClassType, class BaseType, class FieldType> FieldFromBase(const char* name, FieldType BaseType::* address)
```

Create a field from a base class member. This can be used if you want to serialize a base class member without registering and serializing a whole base class, to decouple the serialized class from its base.
- **name** - The name to store the field as. Field names for the same class must be unique. Matching the member name isn't required.
- **address** - The address of the field to store, as a pointer to member or offset from the start of an instance of `BaseType`. If a pointer to member is used, all type information is inferred.

Example Registering a class for serialization

The following is an example from the Lumberyard Asset Processor code, demonstrating how a class can be registered for serialization.

```cpp
if (AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
{
    serializeContext->Class<AssetBuilderDesc>()
        ->Version(2)
        ->Field("Flags", &AssetBuilderDesc::m_flags)
        ->Field("Name", &AssetBuilderDesc::m_name)
        ->Field("Patterns", &AssetBuilderDesc::m_patterns)
        ->Field("BusId", &AssetBuilderDesc::m_busId)
        ->Field("Version", &AssetBuilderDesc::m_version)
        ->Field("AnalysisFingerprint", &AssetBuilderDesc::m_analysisFingerprint);
}
```

Register enums

Enums are registered on a serialization context with the `AZ::SerializeContext::Enum<T>()` method, using the type `T` to determine which enum to register. In order to be serialized, the enum **must** be a specialization of the `AzTypeInfo` using the `AZ_TYPE_INFO_SPECIALIZE()` macro. The `AZ::SerializeContext::Enum<T>()` method returns an `AZ::SerializeContext::EnumBuilder` object, which is used to store version and value information for the enum.

```cpp
AZ::SerializeContext::EnumBuilder
```

Version(unsigned int version, VersionConverter converter = nullptr)

Sets version information for the serialization.
- **version** – The version of the enum. Unlike classes, this doesn't need to change whenever an enum's definition is updated, and is mostly for conversion purposes.
• converter – Converter function which translates older versions of the enum to the provided version.

```cpp
template<class EnumType>
Value(const char* name, EnumType value)
```

Tags an enum value for serialization as part of the enum's information.

• name - The name to store the value as. Field names for the same enum must be unique. Matching the internal value name isn't required.

• value - The associated value to store for the enum. If this is a value associated with the enum, all type information is inferred.

**Important**
If you're serializing a member from a class with an enum type, that enum **must** be registered with the serializer.

### Example Registering an enum for serialization

The following is an example showing a sample declaration of an enum, and a short function that can be called to register it with the serialization system.

```cpp
eenum ExampleEnum : uint8_t
{
    BaseValue = 0,
    Flag1 = 0x1,
    Flag2 = 0x2,
    Flag3 = 0x4,
    Flag4 = 0x8
}

// AzTypeInfo must be set from within the AZ namespace
namespace AZ
{
    AZ_TYPE_INFO_SPECIALIZE(ExampleEnum, "{7ebef8a5-b40d-4a9a-8511-162da1dc02f0}");
}

void registerEnum(AZ::SerializeContext* context)
{
    if (AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
    {
        context->Enum<ExampleEnum>()
            ->Value("Base", ExampleEnum::BaseValue)
            ->Value("Flag1", ExampleEnum::Flag1)
            ->Value("Flag2", ExampleEnum::Flag2)
            ->Value("Flag3", ExampleEnum::Flag3)
            ->Value("Flag4", ExampleEnum::Flag4)
    }
}
```

## Object serialization to JSON format

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Starting with Lumberyard version 1.25, objects can be serialized out to the JSON format. JSON is a machine-readable format that's also designed to be easy for humans to read and edit, so that...
contributors can make small changes to data without requiring specialized tools or a detailed XML schema. Many programming languages also offer robust and easy to use support for JSON data, making the development of build and design tools for supporting your project outside of the Lumberyard editor easy.

In Lumberyard, using the JSON serialization format also offers the following benefits:

- **Minimal serialization** – Only data relevant to the object is serialized. Default values and information that can be retrieved at runtime aren’t stored. This reduces object size and makes manual editing easier.

- **Best-effort deserialization** – To complement the minimal serialization, the engine applies as much data as possible when deserializing. If a field is missing from the JSON input, default values are used. Fields in the JSON which aren’t supported by the object are ignored. This makes working with JSON less error-prone than XML, which requires a rigid structure.

- **Forgiving** – Because of the forgiving nature of JSON, serialization and deserialization can proceed even if it finds missing data while emitting a warning (and optionally reporting an error.) This extends to specialized data types used by Lumberyard such as vectors: If a 3D vector contains a 4th coordinate, it's simply ignored and a warning is reported. This makes manual editing and custom tooling much safer.

- **Easier to use with source control** – Source control systems can have difficulty with merging or generating useful diffs with XML files. Because of JSON's flexibility and limited format, it's easier to work with JSON files in your source control system as text files.

This section of the documentation covers serialization and deserialization of data in the JSON format, and how Lumberyard data types are mapped to JSON data types.

**Topics**

- Serialize and deserialize JSON objects (p. 1981)
- Data types in serialized JSON (p. 1987)

### Serialize and deserialize JSON objects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Once a class has been registered with a serialization context (p. 1978) objects of that class can be serialized and deserialized. Objects are serialized to JSON with the `AZ::JsonSerialization::Store()` function, and deserialized from JSON with `AZ::JsonSerialization::Load()`.

This article includes reference for these methods, examples of using serialization and deserialization, and how to interpret result codes from JSON the serializer. For information on how specific types are serialized, see Data types in serialized JSON (p. 1987).

### Serialization

Serialization into JSON is done with the static `AZ::JsonSerialization::Store()` method. This method has several overloads, depending on how you want the object to be serialized and what information is available at the time of serialization. By default, the global serialization context is used.
AZ::JsonSerialization::Store() overloads

template<typename T> static AZ::JsonSerializationResult::ResultCode
AZ::JsonSerialization::Store(rapidjson::Value& output,
rapidjson::Document::AllocatorType& allocator, const T& object,
AZ::JsonSerializerSettings settings = AZ::JsonSerializerSettings());

- output – The RapidJSON document or value to write to. Objects can be serialized at an arbitrary
  point in a JSON document by providing the appropriate value.
- allocator – The memory allocator used by RapidJSON.
- object – The object to serialize. This object's class must be registered with the provided
  serialization context.

When serializing, a second object of type T is created from the default constructor (if possible) to
provide default values.

- settings – Configuration for how to treat the serialization. If not provided, the default settings
  are used.

template<typename T> static AZ::JsonSerializationResult::ResultCode
AZ::JsonSerialization::Store(rapidjson::Value& output,
rapidjson::Document::AllocatorType& allocator, const T& object,
const T& defaultObject, AZ::JsonSerializerSettings settings =
AZ::JsonSerializerSettings());

- output – The RapidJSON document or value to write to. Objects can be serialized at an arbitrary
  point in a JSON document by providing the appropriate value.
- allocator – The memory allocator used by RapidJSON.
- object – The object to serialize. This object's class must be registered with the provided
  serialization context.
- defaultObject - An object providing the values to treat as the defaults during serialization.
  Any members of object with values that don't match defaultObject are guaranteed to be
  serialized.
- settings – Configuration for how to treat the serialization. If not provided, the default settings
  are used, except that default values will be stored in the output.

static AZ::JsonSerializationResult::ResultCode
AZ::JsonSerialization::Store(rapidjson::Value& output,
rapidjson::Document::AllocatorType& allocator, const void* object, const void*
defaultObject, const AZ::Uuid& objectType, AZ::JsonSerializerSettings settings
= AZ::JsonSerializerSettings());

- output – The RapidJSON document or value to write to. Objects can be serialized at an arbitrary
  point in a JSON document by providing the appropriate value.
- allocator – The memory allocator used by RapidJSON.
- object – The object to serialize, as anonymous data.
- defaultObject - An object providing the values to treat as the defaults during serialization.
  Any members of object with values that don't match defaultObject are guaranteed to be
  serialized. If a null pointer is passed as the default object, a temporary default may be created
during serialization.
- objectType - The UUID registered with the Lumberyard runtime representing the class for
  the provided object. The class represented by this UUID must be registered with the provided
  serialization context.
- settings – Configuration for how to treat the serialization. If not provided, the default settings
  are used, except that default values will be stored in the output provided defaultObject is not
  null.
**AZ::JsonSerializerSettings**

The behavior of the `AZ::JsonSerialization::Store()` methods can be controlled by setting an instance of `AZ::JsonSerializerSettings` as the settings argument.

```cpp
bool m_keepDefaults
```

If `true`, then defaults are written to the JSON value when serialized.

**Default:** `false`

```cpp
AZ::SerializeContext* m_serializeContext
```

The serialization context to query for information about how to serialize the provided object.

**Default:** The serialization context retrieved from the `ComponentApplicationBus` event bus.

```cpp
AZ::JsonSerializationResult::JsonIssueCallback m_reporting
```

Callback method invoked when an error is encountered during serialization. This function has no access to the object being serialized or the JSON value being written to, but can be used for altering result codes.

**Default:** The default issue reporter, which logs warnings and errors encountered in serialization.

```cpp
AZ::JsonRegistrationContext* m_registrationContext
```

JSON registration context. For examples of how to use a custom JSON context, see the source code.

**Default:** The registration context retrieved from the event bus.

### Example Serialization example

The following is a short example demonstrating how to serialize a simple class. Details regarding registering the class with the serialization context are omitted.

```cpp
class SerializableClass
{
    double m_var1;
    int m_var2;
}

// ... Register with serialization context

SerializableClass instance;
instance.m_var1 = 42.0;
instance.m_var2 = 88;

rapidjson::Document document;
AZ::JsonSerializationResult::ResultCode result = AZ::JsonSerialization::Store(document, document.GetAllocator(), instance);
if (result.GetProcessing() == AZ::JsonSerializationResult::Processing::Halted)
{
    AZ_Warning("Serialization", false,
                "Unable to fully serialize SerializableClass to json because %s.",
                result.ToString().c_str());
}

rapidjson::StringBuffer buffer;
rapidjson::Writer<decltype(buffer)> writer(buffer);
document.Accept(writer);
AZ_TracePrintf("Serialization", "SerializableClass as Json:\n%s", buffer.GetString());
```
Deserialization

Deserialization from JSON into an object is done with the static
AZ::JsonSerialization::Load() method. This method has two overloads - one for use when an
instance of the deserialized object type is available, and the other using void* and RTTI information.

AZ::JsonSerialization::Load() overloads

template<typename T> static AZ::JsonSerializationResult::ResultCode
AZ::JsonSerialization::Load(T& object, const rapidjson::Value& root,
AZ::JsonDeserializerSettings settings = AZ::JsonDeserializerSettings{});
• object - The object to load data into.
• root - The root of the JSON tree to deserialize from. This is normally a full JSON document, but
can be any JSON value that will deserialize correctly to type T.
• settings - Configuration for how to treat deserialization.

static AZ::JsonSerializationResult::ResultCode
AZ::JsonSerialization::Load(void* object, const AZ::Uuid& objectType,
const rapidjson::Value& root, AZ::JsonDeserializerSettings settings =
AZ::JsonDeserializerSettings{});
• object - A pointer to memory allocated as an object matching the type registered for
objectType.
• objectType - The UUID registered with the Lumberyard runtime representing the class for
the provided object. The class represented by this UUID must be registered with the provided
serialization context.
• root - The root of the JSON tree to deserialize from. This is normally a full JSON document, but
can be any JSON value that will deserialize correctly to the type identified by objectType.
• settings - Configuration for how to treat deserialization.

AZ::JsonDeserializerSettings

The behavior of the AZ::JsonSerialization::Load() methods can be controlled by setting an
instance of AZ::JsonDeserializerSettings as the settings argument.

bool m_clearContainers

If true, then container members of the target object to deserialize into are cleared before
beginning deserialization. This option has no effect for classes with a fixed layout.

Default: false
AZ::SerializeContext* m_serializeContext

The serialization context to query for information about how to deserialize to the object.

Default: The global serialization context retrieved from the ComponentApplicationBus event bus.
AZ::JsonSerializationResult::JsonIssueCallback m_reporting

Callback method invoked when an error is encountered during deserialization. This function has no
access to the object being serialized or the JSON value being written to, but can be used for altering
result codes.

Default: The default issue reporter, which logs warnings and errors encountered in deserialization.
AZ::JsonRegistrationContext* m_registrationContext

JSON registration context. For examples of how to use a custom JSON context, see the source code.

Default: The global registration context retrieved from the event bus.
Example Deserialization example

The following is a short example demonstrating how to deserialize. Details regarding how to load JSON data into memory and registering with the serialization context are omitted.

```cpp
rapidjson::Document document;  // ...read json document from file.
SerializableClass instance;
AZ::JsonSerializationResult::ResultCode result = AZ::JsonSerialization::Load(instance, document);
if (result.GetProcessing() == AZ::JsonSerializationResult::Processing::Halted)
{
    AZ_Warning("Deserialization", false,
        "Unable to fully deserialize SerializableClass from json because %s.",
        result.ToString().c_str());
}
```

Result codes

The JSON serializer uses the `AZ::JsonSerializationResult::ResultCode` type to report errors, warnings, and successful serialization and deserialization of objects. Result codes are broken down into three parts: task, processing result, and final outcome. These values can be obtained with the `GetTask()`, `GetProcessing()`, and `GetOutcome()` methods respectively. When checking results, you should first start with the processing to see if the task was successfully completed or if an error was encountered. For non-Success values, the task indicates where in processing the error was encountered, and the outcome reflects why the failure occurred.

To write serialization results to a string, use the `AZ::JsonSerializationResult::ResultCode::AppendToString()` and `AZ::JsonSerializationResult::ResultCode::ToString()`, or `AZ::JsonSerializationResult::ResultCode::ToOSString()` methods.

Processing results

Completed

- Processing completed successfully.

Altered

- Processing completed after encountering an error. Error recovery was performed, so the input and output will not necessarily match.

PartialAlter

- Processing completed after encountering multiple errors, and was able to perform error recovery by making multiple alterations.

Halted

- Processing failed and was unable to complete. This indicates an unrecoverable error or other serious failure.

Tasks

RetrieveInfo

- Retrieve information from the system, such as querying a serialization context.

CreateDefault

- Creation of a default instance to use to provide default values during processing.
Convert
Type conversion between values.
Clear
Clearing a field or value.
ReadField
Reading a field from JSON to write to an object value.
WriteValue
Writing from an object value to a JSON field.
Merge
Merging two JSON values or documents together.
CreatePatch
Creation of a patch to transform one JSON value to another.

Outcomes
Success
The task completed successfully.
Skipped
The task skipped a field or value.
PartialSkip
The task skipped one or more fields while processing a JSON object or array.
DefaultsUsed
Task completed using only default values.
PartialDefaults
Task completed using defaults for some values.
Unavailable
Task tried to use space which isn't available.
Unsupported
An unsupported action was requested while performing the task.
TypeMismatch
The source and target were unrelated, and no type conversion could be performed.
TestFailed
A test check against a value failed.
Missing
A required field or value was missing.
Invalid
A field or element contained an invalid value.
Unknown
Unknown information was encountered during the task.
**Catastrophic**

An unidentifiable or unknown catastrophic error occurred during the task.

**Data types in serialized JSON**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In addition to primitive C++ types that map directly to JSON types, Lumberyard supports serializing many AZStd library objects. JSON output and deserialized objects are entirely deterministic based on the appropriate C++ type. For some more information on how members are registered and how their types are determined through the reflection system, see JSON serialization (p. 1981).

This topic is a reference of the types supported by Lumberyard serialization and deserialization, how the serializer maps them by default, and information on how JSON types are coerced back to C++ types.

**Primitives**

The primitive types used in serialization are *booleans*, *integers*, *floating point numbers*, and *strings*. These C++ types map naturally to the native JSON boolean, number, and string types. The only string types supported for serialization are AZStd::string and OSString.

Deserialization can be conducted from any primitive JSON type to these C++ types as follows:

**Basic type mappings**

<table>
<thead>
<tr>
<th>C++ Type</th>
<th>JSON boolean</th>
<th>JSON number</th>
<th>JSON string</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Direct mapping.</td>
<td>0 maps to false, all other values are true.</td>
<td>Case-insensitive comparison against &quot;true&quot; and &quot;false&quot; to map to the respective C++ values.</td>
</tr>
<tr>
<td>Integer types</td>
<td>True maps to 1, False to 0.</td>
<td>Direct mapping, where floating point values are truncated.</td>
<td>Tries to extract a 64-bit integer and convert to the target integer type.</td>
</tr>
<tr>
<td>Floating point types</td>
<td>True maps to 1, False to 0.</td>
<td>Direct mapping.</td>
<td>Tries to extract a 64-bit floating point and convert it to the target type.</td>
</tr>
<tr>
<td>Strings (AZStd::string and OSString only)</td>
<td>True is converted to &quot;True&quot;, and False to &quot;False&quot;.</td>
<td>String representation of the number.</td>
<td>Direct mapping.</td>
</tr>
</tbody>
</table>

**Pointers**

Pointers and smart pointer types are serialized out to the type that they point to as JSON values, following all of the rules for that type. For pointers to C++ objects, this means that the pointed-to object
must be registered with the serialization context being used. While most types can be safely serialized without any metadata, for derived classes, an additional $\texttt{type}$ key is serialized into the JSON object with the class' name. If there could possibly be a conflict in the class name, fully namespaced names or a class' UUID in the Lumberyard runtime can be used.

When deserializing, the $\texttt{type}$ value is used as a hint to determine which object to reconstruct if necessary. Otherwise, only the C++ type of the member is inspected. Deserialization proceeds memberwise, using the field mapping that was registered with the serialization context to map JSON object keys to C++ members.

**Enums**

In order to serialize enums, they need to be registered (p. 1979) with the same serialization context as the class being serialized. How values from an enum are serialized and deserialized depend on the details of the enum registration.

- **Enum value matches the value of a registered field** – The value is serialized as a string containing the field name.
- **Enum value can't be represented as ORed registered values** – The value is serialized as an integer.
- **Enum value can be represented as ORed registered values** – The value is serialized to an array containing strings of the field names that were ORed together.
- **Enum value can be represented as a mix of ORed registered values and unregistered values** – The value is serialized to an array containing the strings of the appropriate field names and a single integer that when ORed together, produce the serialized value.

**Example Serializing an enum**

These rules are best illustrated with a simple example to show an enum definition, its registration with the serialization context, and the serialization output for various values.

```c
enum ExampleEnum : uint8_t
{
    Flag1 = 1,
    Flag2 = 2,
    Flag3 = 4,
    Flag4 = 8,
    Flag5 = Flag2 | Flag3
}

//------
// Registration with SerializedContext
serializeContext->Enum<ExampleEnum>()
   ->Value("Flag1", ExampleEnum::Flag1)
   ->Value("Flag2", ExampleEnum::Flag2)
   ->Value("Flag4", ExampleEnum::Flag4)
   ->Value("Flag2Flag3Combo", ExampleEnum::Flag5)

//------

// ExampleEnum initialization with int
ExampleEnum(0) // Serializes as 0
ExampleEnum(1) // Serializes as "Flag1"
ExampleEnum(-1) // Serializes as 255
ExampleEnum(10) // Serializes as ["Flag2", "Flag4"]

// Declared ExampleEnum values
ExampleEnum::Flag1 // Serializes as "Flag1"
ExampleEnum::Flag3 // Serializes as 4
ExampleEnum::Flag5 // Serializes as "Flag2Flag3Combo"
```
// ORed together values
ExampleEnum::Flag1 | ExampleEnum::Flag4   // Serializes as ["Flag1", "Flag4"]
ExampleEnum::Flag2 | ExampleEnum::Flag3   // Serializes as "Flag2Flag3Combo"
ExampleEnum::Flag1 | ExampleEnum::Flag3 | ExampleEnum::Flag1 // Serializes as ["Flag1", "Flag2Flag3Combo"]
ExampleEnum::Flag4 | 16      // Serializes as ["Flag4", 16]

Deserialization behaves as expected based on these serialization outputs. Strings are first attempted to be mapped to a field name, and then the enum is assigned the field's value. If the string contains something other than a field name, a type conversion to an integer is attempted. Arrays have each element evaluated to convert to the enum field or an integer value, and then the results of those evaluations are ORed to produce the final deserialized value.

Vectors

Serialization supports 2D, 3D, and 4D vectors. An N-dimensional vector is serialized as a JSON array of N floating point numbers, in the order of the X, Y, Z, and W coordinate.

For deserialization, vectors can be read from an array of any length, but will only read up to the number of elements in the target type. If the array has fewer elements than the vector type, those vector components are assigned the default value of 0. Array elements follow the rules of deserialization for the C++ floating point type of the vector elements. Vectors can also be deserialized from JSON objects where the object keys map to the vector component name in a case-insensitive compare. Other keys are ignored, and components with missing keys use the default value.

Containers

Serialization supports a number of containers from AZStd. These include the array, vector, list, set, map, and tuple types.

Array types

The following types are serialized to JSON arrays:

- AZStd::array
- AZStd::fixed_vector
- AZStd::forward_list
- AZStd::list
- AZStd::vector
- AZStd::pair
- AZStd::tuple
- AZStd::set - Values in the serialized array are sorted
- AZStd::unordered_set - Values in the serialized array are sorted

Deserialization of a JSON array to a C++ array, list, vector, set, pair, or tuple type is a direct element-by-element conversion. The types of each array element are converted to the target container's value type according to the other JSON deserialization rules. Missing elements map to the default for the container's value type, and additional elements are ignored. Types other than JSON arrays will result in a conversion error when attempting to deserialize to one of these C++ types.

Map types

The serialization of map types (AZStd::map, AZStd::unordered_map, AZStd::unordered_multimap) depends on the map's key type. Maps with a serializable string type
(AZStd::string or OSString) as the key are serialized directly to JSON objects with equivalent key/value pairs. Maps with any other key type are serialized to JSON arrays, where every element is a key/value pair. For example, the map AZStd::map<uint8_t, uint8_t> = {0:1, 2:3} serializes to the JSON array ["Key": 0, "Value": 1], ["Key": 2, "Value": 3].

**Miscellaneous types**

Other internal types supported by JSON serialization are *UUID* and *color*.

AZ::Uuid are mapped to JSON strings by AZ::Uuid::ToString(), and strings are converted back to UUIDs from AZ::Uuid::CreateString().

Color types are serialized to JSON arrays containing 3 float values for RGB colors, and 4 values for RGBA. JSON arrays of 3 or 4 values can be deserialized directly back to RGB or RGBA, but JSON objects can also be deserialized to colors. For an object to be deserialized to a color, it must have exactly one key of the following names, with an equivalent value type.

**Color deserialization from JSON objects**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>Array of 3 floating point elements.</td>
<td>{ &quot;RGB&quot; : [1.0, 0.3, 0.2] }</td>
</tr>
<tr>
<td>RGBA</td>
<td>Array of 4 floating point elements.</td>
<td>{ &quot;RGBA&quot; : [1.0, 0.3, 0.2, 0.8] }</td>
</tr>
<tr>
<td>RGB8</td>
<td>Array of 3 integer elements with values in the range [0, 255].</td>
<td>{ &quot;RGB8&quot; : [255, 77, 51] }</td>
</tr>
<tr>
<td>RGBA8</td>
<td>Array of 4 integer elements with values in the range [0, 255].</td>
<td>{ &quot;RGBA8&quot; : [255, 77, 51, 204] }</td>
</tr>
<tr>
<td>HEX</td>
<td>A string containing 3 8-bit hex values representing the colors.</td>
<td>{ &quot;HEX&quot; : &quot;FF4D33&quot; }</td>
</tr>
<tr>
<td>HEXA</td>
<td>A string containing 4 8-bit hex values representing the colors and alpha channel.</td>
<td>{ &quot;HEXA&quot; : &quot;FF4D33CC&quot; }</td>
</tr>
</tbody>
</table>

**Using Lumberyard Networking**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard's GridMate networking system is designed for efficient bandwidth usage and low-latency communications. You can synchronize objects over the network with GridMate's replica framework. GridMate's session management integrates with major online console services and lets you handle peer-to-peer and client-server topologies with host migration. GridMate also supports in-game achievements, leaderboards, and cloud-based saved games through third-party social services such as Steam. For an example of how to set up a multiplayer project, see Multiplayer Sample (p. 155).
This section discusses the various components of, and setup requirements for, your Amazon Lumberyard game networking environment. For information about a diagnostic tool for networking, seeProfiler (p. 1920).

**Topics**

- Tutorial: Getting Started with Multiplayer (p. 1991)
- Overview (p. 2008)
- Synchronizing Game State Using Components (p. 2049)
- Synchronizing Game State Using Scripts (p. 2067)
- Using Encryption (p. 2067)
- Controlling Bandwidth Usage (p. 2072)
- Using Bit Packing in Lumberyard Networking (p. 2074)
- Setting up a Lobby (p. 2076)
- Creating Dedicated Servers (p. 2076)
- Using the TCP Stream Driver (p. 2077)
- Using GridMate for Large-Scale Worlds (p. 2080)
- Using Amazon GameLift (p. 2088)
- Useful Console Commands (p. 2091)
- Using Multiplayer Analytics (p. 2092)

**Tutorial: Getting Started with Multiplayer**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This tutorial walks you through the steps to create a simple multiplayer test game level. These steps include binding an entity to the network and connecting a client to the host. At the end of the tutorial, you should have a level with a simple network bound entity that is ready for a multiplayer game.

This tutorial guides you through the following tasks:

- Create a level and add entities.
- Bind an entity's **Transform** component to the network.
- Connect a client to the server and verify network replication.

**Prerequisites**

This tutorial assumes the following:

- You installed Amazon Lumberyard.
- You created a game project. For more information, see Creating and Switching Game Projects (p. 44).
- Your game project has the **Multiplayer** gem. You can enable the gem in Lumberyard's Project Configurator (p. 43). After enabling the gem, build your project (p. 61).

**Note**

This tutorial uses Visual Studio 2017.

**Topics**
Step 1: Creating a Level and Adding a Sphere and a Box

Your first step is to create a level and prepare a simple sphere and box shape so that you can test Lumberyard's networking features.

To create a level, sphere, and box

1. In the Lumberyard Project Configurator, select a project that has the Multiplayer gem enabled, and then click Set as default.
2. Open Lumberyard Editor, create a level, and enter a name.
3. In the Lumberyard Editor viewport, right-click and choose Create entity.
4. With the entity selected, click Tools, Entity Inspector. Use the Entity Inspector to name the entity CameraEntity.
5. Click Add Component.
6. Select the Camera component to attach it to the entity.
7. In the Lumberyard Editor viewport, right-click and choose **Create entity**.
8. With the entity selected, use the **Entity Inspector** to name the entity **SphereEntity**.
9. In the **Entity Inspector**, click **Add Component**, **Rendering**, **Mesh** to attach a **Mesh** component to the **SphereEntity**.
10. In the **Mesh** component, click the (...) icon next to **Mesh asset**.
11. In the **Pick Static Mesh** window, expand **Engine, Objects, default**, and select **primitive_sphere.cgf**.
12. Click OK.
13. In the viewport, right-click and choose Create entity.
14. With the entity selected, use the Entity Inspector to name the entity BoxEntity.
15. In the Entity Inspector, click Add Component, Rendering, Mesh to attach a Mesh component to the BoxEntity.
16. In the Mesh component, click the (...) icon next to Mesh asset.
17. In the Pick Static Mesh window, under Engine, Objects, default, select primitive_cube.cgf.
18. Click OK.
19. In the viewport, select the SphereEntity. In the Entity Inspector, click Add Component, Physics, Rigid Body Physics.
20. Select the BoxEntity. In the Entity Inspector, click Add Component, Physics, Rigid Body Physics.

21. Select the SphereEntity. In the Entity Inspector, click Add Component, Physics, Mesh Collider.
22. Select the **BoxEntity**. In the **Entity Inspector**, click **Add Component**, **Physics**, **Mesh Collider**.

23. In the viewport, move the sphere and box entities above the plane so that they have room to fall.

   **To move the sphere and box entities above the plane**

   a. In the viewport, select the entity.

   b. On the **EditMode** toolbar, choose the **Move** icon.

   c. On the **EditMode** toolbar, choose the **Constrain to Z Axis** icon.

   d. In the viewport, drag the item up vertically.

24. In the **Entity Inspector**, **Rigid Body Physics** component, ensure that **At rest initially** is unchecked. This allows the sphere and box to begin simulating after the level is loaded.
You now have two simple component entities with rigid body physics in the level that you created.
Step 2: Binding Sphere Transform Components to the Network

After you create the initial level with the sphere and the box, you bind the sphere's Transform component to the network. This allows the sphere's changes to be replicated over the network to clients.

To bind the sphere's transform to the network

1. Select the sphere entity.
2. In the Entity Inspector, click Add Component, Networking, Network Binding. Adding the NetBinding component to the entity allows the host to replicate the Transform component of the sphere to all clients.
   
   ![Entity Inspector](image)

3. With the sphere entity selected, in the Transform component, ensure that the Network Sync, Sync to replicas property is enabled.
You have now created a server authoritative sphere entity that enables changes to the sphere to replicate over the network. However, you didn't bind the box to the network, so changes in the box will remain unreplicated.

**Step 3: Connecting a Client to the Server**

This step shows you how to connect a client to the server instance and then observe your networked sphere in action.

**To connect a client game to the host game**

1. In Lumberyard Editor, choose Game, Export to Engine, or press Ctrl+E to export your level.
2. Run the game launcher from the Bin directory that you are using. The name of your launcher is `your_game_project_name`Launcher.exe.
   
   For Visual Studio 2017, the directory is `lumberyard_version\dev\Bin64vc141\`.  
   
   For Visual Studio 2019, the directory is `lumberyard_version\dev\Bin64vc142\`.  
3. Press the `key (above the Tab key) to open the console.
4. Run the command `map MultiplayerTutorial` where `MultiplayerTutorial` is the name of the level to load.
5. Press the ` key to open the console. Run the command `mphost` to make your client host a network session.

6. Move the first launcher window to one side so that you will be able to see the second launcher window. Use the launcher again to open a second instance of the game. Press the ` key to open the second console.

7. Run the command `sv_port 0` to set the client port to 0 (the ephemeral port).

   **Note**
   On a single computer, only one process is allowed to bind to a particular port. Therefore, to run more than one game process on the same computer (as in this multiplayer sample), you
must use ephemeral ports. The sv_port console variable defines the UDP port on the local machine for the multiplayer sample, and the setting of 0 specifies the ephemeral port. This allows two clients on the same computer to talk to each other.

8. Run the command `mpjoin` to join to the host game. You should see the sphere synchronized by location on the client. However, the box will be desynchronized and have different locations on the client and host.

You have successfully created a simple networked level. You can now use the **Network Binding** component to connect clients to servers and synchronize transforms of entities.

### Related Tasks and Tutorials

You have created a simple networking sample to see the effects of networking in Lumberyard. See the following to learn more about what else you can add to your game:

- Configuring the Multiplayer Sample for Amazon GameLift (p. 2003)
- Using Amazon GameLift

### Configuring the Multiplayer Sample for Amazon GameLift

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To prepare the multiplayer sample for use with Amazon GameLift, follow the required procedures for server-side and client-side configuration.

### Server-Side Configuration

On the server side, overwrite the GridMate::OnSessionStarted() handler. In the handler, synchronize the session state and load the corresponding map if the CVAR sv_map is set in the Multiplayer::Utils::SynchronizeSessionState() function.
The following example shows code for server-side configuration.

```cpp
void GameManager::OnSessionCreated(GridMate::GridSession* session)
{
    m_gameSession = session;
    if (m_gameSession)
    {
        if (m_gameSession->IsHost())
        {
            if (gEnv->IsDedicated())
            {
                Multiplayer::Utils::SynchronizeSessionState(m_gameSession);
            }
        }
    }
}
```

Set the following server CVARs as needed:

gamelift_flexmatch_enable

gamelift_flexmatch_onplayerremoved_enable

gamelift_flexmatch_start_delay

For more details on these server CVARs provided by the Multiplayer Gem, see Server CVars (p. 1178).

**Client-Side Configuration**

On the client side, you must configure the following CVARs:

sv_port

sv_map

gamelift_aws_access_key

gamelift_aws_secret_key

gamelift_fleet_id or gamelift_alias_id

gamelift_queue_name

gamelift_end_point

gamelift_playerid

gamelift_matchmaking_config_name

You can set these CVARs with a console command or with the multiplayer sample user interface. For a list of all the supported CVARs in the Multiplayer Gem, see Multiplayer Gem CVars (p. 1176).

To use CVARs to set the client side configuration, enter the following console command.

```
+sv_port 33435 +gamelift_fleet_id <fleet> +gamelift_aws_access_key <aws access key>
+gamelift_aws_secret_key <aws secret key>
```

**To use the multiplayer sample user interface to configure Amazon GameLift**

1. By default, the multiplayer sample loads the Game Lobby map. To add or modify the CVARs, choose Amazon GameLift.
2. Fill in the details of the GameLift configuration and choose **Connect**.

3. To **create** a game session and join automatically, select the **Create Server** radio button and do the following:
   
i. Specify the **Server Name** and the **Map** (`sv_map`) to load.
   
ii. Enter the **Queue Name**, **Fleet ID**, or **Alias ID**.
   
iii. Choose the **Create Server** button.
4. To search for and join an active game session, select the Join Server radio button and do the following:
   
i. Specify the Queue Name, Fleet ID, or Alias ID.
ii. Choose Refresh. Active game sessions will be listed.
iii. Select a session and choose Join.

5. To use matchmaking to get placed in a game session, select the FlexMatch radio button and do the following:
i. Specify the matchmaking **Config Name**.
ii. Choose **Start Matchmaking**.

![GameLift Lobby](image)

**Create an Amazon GameLift Package**

To create a Amazon GameLift package, complete the following steps.

**To create a Amazon GameLift package**

1. Before you create an Amazon GameLift package, do the following:
   - Compile game assets
   - Build the Lumberyard executable
2. Run the following commands to create the Amazon GameLift package:

   ```bash
   mkdir GameLiftPackageWindows
   cp -r MultiplayerSample_pc_Paks_Dedicated/* GameLiftPackageWindows/
   cp -r Bin64vc141.Dedicated/* GameLiftPackageWindows/
   ```
3. Copy the corresponding Visual Studio redistributable into the `GameLiftPackageWindows` folder.
4. Verify the dedicated server runs from the `GameLiftPackageWindows` folder.
5. Upload your build and create a fleet from the Amazon GameLift console. For more information, see [Uploading Your Game to Amazon GameLift](#).
Secured Connection (Not Amazon GameLift Specific)

Amazon GameLift uses the OpenSSL-based secure socket driver to create a secured connection. However, instead of verifying the server, the secure socket driver can verify the client.

To enable a secured connection, make the following change to the `game.cfg` file:

\[
\text{gm_netsec_enable} = 1
\]

If client verification is needed, make the following change to the `game.cfg` file:

\[
\text{gm_netsec_verify_client} = 1
\]

**Note**

By default, the certificate and private key are loaded from the `multiplayersample.cert.pem` file (shared by the certificate and CA root) and from the `multiplayersample.key.pem` file. To specify different files, use the `gm_netsec_certificate` and `gm_netsec_private_key` CVARs.

Overview

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard enables multiplayer functionality through the following software layers:

- AZFramework
- Netbinding
- GridMate
  - Replica
  - Session
  - Carrier
  - Driver

These layers are illustrated in the following diagram.
NetBinding

The network binding API of the AzFramework library provides a way for components to synchronize their state over the network. The API is implemented on top of GridMate replicas. A special NetBindingComponent is responsible for the actual binding process, so entities that need to be synchronized need to have a NetBindingComponent added to them. When a game enters a multiplayer session, the NetBindingComponent collects replica chunks from the NetBindables on the entity, and adds them to a replica primary.
GridMate

GridMate is a library that enables you to easily add online features to your games on multiple operating systems and devices. The GridMate API library has two general categories: network synchronization and online systems. Each API is designed to be modular and extensible. Services can be enabled independently of each other, and different implementations can be provided for each API. Optional features are implemented as plugins for ease of customization. GridMate is built on top of Lumberyard's AzCore library. Service APIs are implemented using EBuses (AzCore's implementation of signal/slots) to improve modularity and extensibility. All GridMate allocations are piped through two specific allocators: GridMateAllocatorMP is used for allocations from the network synchronization APIs. GridMateAllocator is used for all other allocations, such as those from the online APIs and core system allocations. GridMate also supports debugging through AzCore's Driller framework. All network and replication events are reported and can be captured for logging and debugging purposes.

Replica

GridMate uses a single-primary replication model. For each replica, one node in the session owns the primary copy, and everyone else has a proxy copy. Replicas can be individually migrated from node to node at any time.

At the core of GridMate's replication model is the replica. Replicas, along with the chunks, datasets and RPCs that make up the replica, provide a mechanism for capturing and propagating the game state. Replicas also serve as the point of interaction for external game systems. Replicas can be owned by any node in the network and can be migrated to whichever node that can process them most efficiently.

Each node in the replication network runs a local instance of replica manager. As a node establishes connections to other nodes, it adds them to the replica manager as peers. This builds out its replication network.

One important design element of GridMate replicas is the broadcast nature of the system. Many replication systems allow users to specify replication targets directly, either per replica or per update. This attempt to enable bandwidth optimizations is error prone and puts the implementation burden on gameplay programmers who are often less familiar with network desynchronization issues. Instead, GridMate's approach follows the rule "when something happens, it happens for everyone".

Session

The session service is responsible for managing and maintaining the connectivity required to other members in a game session. GridMate's session service consists of a simple matchmaking API to facilitate integration with existing matchmaking services, and a session implementation that supports three topologies: P2P full mesh, client/server and a hybrid mode that consists of a full mesh network connected to a client/server network. Host migration is available when using full mesh topology. Host migration is a multi-step process that begins as nodes lose connectivity to the session host. The first step is host election: as nodes disconnect from the host, they broadcast a request for a new host election, and go through a series of voting rounds, until a majority is reached or the election process times out. The new host(s) then starts the migration process, dropping problematic connections and migrating replicas until the session is stable again before resuming normal operations. During this time the connection graph can be very unstable, and a variety of steps are taken to improve success rate.

Carrier

GridMate's carrier implementation provides reliable and unreliable messaging. Messages are sent over a channel. Each channel represents an independent stream of messages. Reliable and unreliable messages can be sent over the same channel. Within a channel, message delivery is always ordered, and out-of-order unreliable messages are always discarded. GridMate supports multiple channels to compartmentalize the effect of packet losses and reordering. GridMate provides separate dedicated channels for replication and voice chat traffic. To minimize impact to and from the game thread,
the current carrier implementation performs network sends and receives from a separate IO thread. Decoupling sends and receives into separate threads and incorporating epoll/IOCP is planned. The carrier API provides hooks for congestion control, connection handshakes and network simulators. Users can use the default implementations in GridMate or provide their own custom implementations.

Driver

The driver is the interface for the lowest level of the transport layer. Lumberyard ships with several driver implementations: SocketDriver is a generic socket driver that supports BSD/WinSock/Posix sockets on the corresponding operating systems. The SecureSocketDriver supports encrypted communication through the DTLS protocol by using OpenSSL.

Other GridMate Features

Other GridMate features include:

- **Online Service** - Provides essential user information used by the other APIs.
- **Achievements** - An API for in-game achievements support.
- **Leaderboards** - An API for leaderboard support.
- **Online Storage** - An API for online storage support.

CryNetwork Backward Compatibility (Deprecated)

Lumberyard has a backwards compatibility layer for the deprecated legacy networking system called "CryNetwork". This layer is mostly encapsulated inside the CryNetwork library and exposed through the INetwork interface. The layer is intended only for projects that were built using CryNetwork so that you can transition them to Lumberyard's network technology (NetBinding components and GridMate). Because the CryNetwork backward compatibility API layer uses CPU and bandwidth inefficiently, we strongly recommend that you do not build or release multiplayer games using it.

For information on the compatibility layer, see [CryNetwork Backward Compatibility](#).

Topics

- Networking Architecture (p. 2011)
- Carrier (p. 2014)
- Marshalling (p. 2022)
- Sessions (p. 2025)
- Replicas (p. 2034)
- Replica Manager (p. 2047)

Networking Architecture

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Fundamental Concepts

Lumberyard provides a network layer that supports a wide variety of game types on multiple operating systems and does not restrict game developers to using any particular network topology. You are able
to create games using three network topologies: P2P full mesh, client/server, and a hybrid mode that consists of a full mesh network connected to a client/server network. You can create gameplay objects that are server authoritative, and gameplay objects that are client authoritative.

In this discussion, peer and host have the following meanings:

**Peer** - A network node that is participating in a game session.

**Host** - A special kind of a peer that manages the game session. The host can run on one of the game clients or be a dedicated server.

Synchronization of the states of various networked game objects is achieved through the GridMate replication model. One important design element is the concept of a horizon. GridMate does not maintain a full graph of the replication network at each node. Instead, each node is only aware of the peers that it has a direct connection to; everything else is considered the "horizon". Nodes keep track of which replica updates are arriving from which peer (upstream) only for purposes of routing, so they know where to forward upstream requests and, in the case of hub nodes, where to send them downstream. Basically, if a node receives a request for a replica it doesn't own, it forwards it upstream.

**GridMate Architecture**

The following diagram shows the major components of the GridMate architecture and their relationships.
For more information, see the following pages.
Carrier

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Carrier is GridMate's messaging API. GridMate's reliable UDP implementation supports both reliable and unreliable messages. There is no out-of-order delivery. Out-of-order messages are queued if sent reliably, or discarded if sent unreliably.

The carrier sends messages through channels. The purpose of channels is to separate unrelated traffic, such as game state and voice chat. Message ordering is not enforced across channels.

The carrier API also provides hooks for congestion control and traffic simulation.

Channels and Message Priorities

Messages can be sent on different channels and have different priorities. Message ordering is always maintained between messages with the same priority sent on the same channel.

Channels provide a way to separate unrelated messages so that their ordering does not affect one other. When messages arrive out of order, they are either discarded or queued (and therefore delayed) depending on their reliability. Using different channels prevents unrelated messages from being unnecessarily dropped or delayed. For example, object replication traffic and voice chat traffic can be sent on different channels, so a missing reliable message for object replication would not cause voice chat data to be dropped, and vice versa.

Customizable Classes

You can customize the following classes to implement your own networking features:

- **Driver** - Carrier defers actual network operations to the driver, so different implementations can be provided for different operating systems and devices. This abstraction makes it possible to use OS or device-specific protocols from service providers such as Steam. The default implementation uses UDP and supports IPv4 and IPv6.

- **Simulator** - If a network simulator is present, the carrier passes all inbound and outbound traffic through it so different network conditions can be simulated. One simulator instance can be supplied per carrier instance. The default implementation can simulate different patterns for inbound and/or outbound latency, bandwidth caps, packet loss and packet reordering.

- **Traffic Control** - The traffic control module has two primary functions: provide network statistics and congestion control. Whenever messages are sent or received, they are passed along to the traffic control module so it can update its statistics, and also so it can provide feedback to limit the amount of data being sent. It also decides if messages should be considered lost and resent by the carrier.

CarrierDesc

CarrierDesc is the carrier descriptor. When you create a carrier, you use the CarrierDesc structure to specify the parameters for the current session.
### CarrierDesc Parameters

The following parameters can be supplied during carrier initialization:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_address</td>
<td>const char *</td>
<td>Specifies the local communication address to which the driver will bind. A value of 0 specifies any address. The default is nullptr.</td>
</tr>
<tr>
<td>m_connectionEvaluationThreshold</td>
<td>float</td>
<td>When a disconnection is detected, specifies the threshold at which all other connections are checked using ( m\text{-}connectionTimeoutMS \times m\text{-}connectionEvaluationThreshold ) to see if they are also failing because of a network failure. The default is 0.5f.</td>
</tr>
<tr>
<td>m_connectionTimeoutMS</td>
<td>unsigned int</td>
<td>Determines the time to allow for a connection attempt. The default is 5000 milliseconds.</td>
</tr>
<tr>
<td>m_disconnectDetectionPacketLossThreshold</td>
<td>float</td>
<td>Packet loss percentage threshold. Possible values are from 0.0 to 1.0, where 1.0 is 100 percent. The connection will be dropped after packet loss exceeds the value specified. The default is 0.3f.</td>
</tr>
<tr>
<td>m_disconnectDetectionRttThreshold</td>
<td>float</td>
<td>Specifies the RTT (round-trip time) threshold in milliseconds. The connection is dropped when the measured RTT is greater than the value specified. The default is 500.0f.</td>
</tr>
<tr>
<td>m_driver</td>
<td>class Driver *</td>
<td>Specifies a custom driver implementation. The default is nullptr.</td>
</tr>
<tr>
<td>m_driverIsCrossPlatform</td>
<td>bool</td>
<td>Specifies whether the driver maintains cross-OS and device compatibility. When true, the default driver drops to the most restrictive MTU (maximum transmission unit) across all supported operating systems and devices. The default is false.</td>
</tr>
<tr>
<td>m_driverIsFullPackets</td>
<td>bool</td>
<td>Specifies whether the driver ignores MTU limits. This parameter applies only to socket drivers and local area networks. An internet packet is usually around 1500 bytes. A value of true enables a maximum packet size of 64 KB. These big packets fail on the Internet but typically do not on local networks. The default is false.</td>
</tr>
<tr>
<td>m_driverReceiveBufferSize</td>
<td>unsigned int</td>
<td>Specifies the size of the internal receive buffer that the driver uses. A value of 0 specifies the default buffer size. This parameter can be used only if m_driver == null. The default is 0.</td>
</tr>
<tr>
<td>m_driverSendBufferSize</td>
<td>unsigned int</td>
<td>Specifies the size of the internal send buffer that the driver uses. A value of 0 specifies the default buffer size. This parameter can be used only if m_driver == null. The default is 0.</td>
</tr>
<tr>
<td>m_enableDisconnection</td>
<td>bool</td>
<td>Specifies whether the carrier drops connections when traffic conditions are bad. The default is true.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This parameter should be set to <code>false</code> only when debugging.</td>
</tr>
<tr>
<td>m_familyType</td>
<td>int</td>
<td>Specifies the protocol family that the driver uses. A value of 0 specifies the default family.</td>
</tr>
<tr>
<td>m_port</td>
<td>unsigned int</td>
<td>Specifies the local communication port to which the driver binds. A value of 0 specifies the port assigned by the system.</td>
</tr>
<tr>
<td>m_securityData</td>
<td>const char *</td>
<td>Specifies a pointer to a string with security data. The default is <code>nullptr</code>.</td>
</tr>
<tr>
<td>m_sendBatchPacketCount</td>
<td>int</td>
<td>Specifies the number of full maximum transmission size (MTU) packets to queue before a send event is triggered. You can use this parameter to batch small messages into a single packet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is 0, which means no batching. When <code>m_sendBatchPacketCount</code> is set to 0, all messages immediately trigger a send event on the carrier thread.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To prevent data transmission from stalling, the carrier thread times out at the time specified by the <code>m_threadUpdateTimeMS</code> parameter. When the carrier thread receives a packet, it also checks the send queue for outgoing messages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This parameter is available in Lumberyard 1.16 and later versions.</td>
</tr>
<tr>
<td>m_simulator</td>
<td>class Simulator *</td>
<td>Optionally specifies a simulator through which all network messages are filtered. When specified, the carrier passes all inbound and outbound traffic through the specified simulator so that different network conditions can be simulated. You can specify one simulator instance per carrier instance.</td>
</tr>
<tr>
<td>m_threadCpuID</td>
<td>int</td>
<td>Restricts the carrier thread to a specific CPU core. The values that can be specified are operating system and device dependent. A value of -1 specifies no restriction. The default is -1.</td>
</tr>
<tr>
<td>m_threadInstantResponse</td>
<td>bool</td>
<td>Specifies whether I/O events wake up the carrier thread immediately. The default is <code>false</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Setting this value to <code>true</code> typically uses more bandwidth because messages (especially small messages) are grouped less efficiently.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In Lumberyard 1.16 and later versions, <code>m_threadInstantResponse</code> affects only the reception of carrier thread packets. To send all messages immediately, the <code>m_sendBatchPacketCount</code> parameter must be set to 0 (the default).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>m_threadPriority</td>
<td>int</td>
<td>Specifies the thread priority for the carrier thread. The values that can be specified are operating system and device dependent. A value of −100000 inherits the priority from calling thread. The default is −100000.</td>
</tr>
<tr>
<td>m_threadUpdateTimeMS</td>
<td>int</td>
<td>Specifies, in milliseconds, how often the carrier thread is updated. This parameter is ignored if m_threadInstantResponse is true. Possible values are from 0 through 100. In general, the time interval should be higher than 10 milliseconds. Otherwise, it is more efficient to set m_threadInstantResponse to true. The default is 30 milliseconds.</td>
</tr>
<tr>
<td>m_trafficControl</td>
<td>class</td>
<td>Specifies a custom traffic control implementation that controls traffic flow to all connections and that handles issues like network congestion.</td>
</tr>
<tr>
<td>m_version</td>
<td>VersionType</td>
<td>Specifies the version of Carrier API that is being used. Carriers with mismatching version numbers are not allowed to connect to each other. The default is 1.</td>
</tr>
</tbody>
</table>

**Topics**

- Carrier Message Structure (p. 2017)
- The TCP Stream Driver (p. 2018)
- Using Windows Registered I/O with GridMate (p. 2020)

**Carrier Message Structure**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This topic describes the message structure used by the CarrierImpl networking class found in the Carrier.cpp file in the GridMate source code.

In the following sections, values in parentheses indicate the field's length in bits. For fields with variable length, the value indicates the minimum length.

**Datagram Format**

The overall datagram has the following structure.

```
DgramID (16) | Msg1 (64+) | Msg2 (24+) | ...
```

**Message Format**

The following diagram shows the possible message fields. Only the first two fields are present in every message header. All the other fields are sent only as necessary. In general, ChannelId and NumChunks are rarely sent. SeqNum and RelSeqNum are usually sent once per datagram.

```
Flags (8) | Length (16) | ChannelId (8) | NumChunks (16) | SeqNum (16) | RelSeqNum (16) | Payload (0+)
```
System Messages

Carrier system messages include ACK and ClockSync.

ACK

The ACK system message is used to ACK any received messages and to keep the connection alive. When there is no activity, an ACK containing only the first two fields is sent, otherwise, the actual fields sent vary depending on the pattern being ack'ed. At the very least, LastToAck is sent. If the sequence ack'ed contains gaps, a variable-length bit set is used; otherwise, the first sequence number being ack'ed is included. These possible message formats are shown in the following diagram.

<table>
<thead>
<tr>
<th>MsgId (8)</th>
<th>Flags (8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsgId (8)</td>
<td>Flags (8)</td>
</tr>
<tr>
<td>MsgId (8)</td>
<td>Flags (8)</td>
</tr>
</tbody>
</table>

ClockSync

A ClockSync message is sent about once per second to keep all the clocks in the session in sync. The message format is as follows.

| MsgId (8) | Time (32) |

The TCP Stream Driver

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Starting with Lumberyard version 1.10, the GridMate library offers a TCP socket driver in addition to its existing UDP driver. Unlike UDP, the TCP protocol uses direct connections, which require a listening port and established server endpoint that clients can use. The GridMate library also adds an event bus that detects stream connect and disconnect events for TCP.

TCP Socket Driver Classes

The TCP socket driver has four public classes in the GridMate namespace that handle TCP packet traffic.

**GridMate::StreamSocketDriver**

This socket driver has methods that handle TCP connections.

**Stream Socket Methods**

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish a direct connection to a listening server</td>
<td>ResultCode ConnectTo(const SocketDriverAddressPtr&amp; addr)</td>
</tr>
<tr>
<td>Break an established TCP connection</td>
<td>ResultCode DisconnectFrom(const SocketDriverAddressPtr&amp; addr);</td>
</tr>
<tr>
<td>Start accepting connections to the server</td>
<td>ResultCode StartListen(AZ::s32 backlog);</td>
</tr>
<tr>
<td>Stop the server from accepting connections</td>
<td>ResultCode StopListen();</td>
</tr>
</tbody>
</table>
This call does not shut down the driver. You can start monitoring after the call.

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poll the number of established connections</td>
<td>AZ::u32 GetNumberOfConnections() const;</td>
</tr>
<tr>
<td>Query whether a remote address endpoint is connected</td>
<td>bool IsConnectedTo(const SocketDriverAddressPtr&amp; to) const;</td>
</tr>
<tr>
<td>Query whether the driver is actively listening for new connections</td>
<td>bool IsListening() const;</td>
</tr>
</tbody>
</table>

**GridMate::StreamSocketDriverEventsBus**

The socket driver event bus has methods that detect connection and disconnection events.

**Stream Socket Events Bus Methods**

<table>
<thead>
<tr>
<th>Callback that occurs when</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>The socket driver establishes a connection</td>
<td>virtual void OnConnectionEstablished(const SocketDriverAddress&amp; address);</td>
</tr>
<tr>
<td>The socket driver detects a removed connection</td>
<td>virtual void OnConnectionDisconnected(const SocketDriverAddress&amp; address);</td>
</tr>
</tbody>
</table>

**GridMate::StreamSecureSocketDriver**

The secure stream socket driver has the following method that initializes cryptographic data for the driver. It calls the Initialize() method before setting up the key and/or certificate.

```
ResultCode InitializeSecurity(AZ::s32 familyType, const char* address, AZ::u32 port, AZ::u32 receiveBufferSize, AZ::u32 sendBufferSize, StreamSecureSocketDriverDesc& desc);
```

**GridMate::StreamSecureSocketDriver::StreamSecureSocketDriverDesc**

The secure stream socket driver requires a description structure to set up the cryptographic key, certificates, and options. The description structure has the following members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char* m_privateKeyPEM;</td>
<td>A base64 encoded PEM format private key that is used on the server only.</td>
</tr>
<tr>
<td>const char* m_certificatePEM;</td>
<td>A base64 encoded PEM format certificate. This public certificate encrypts the Transport Layer Security (TLS) handshake.</td>
</tr>
<tr>
<td>const char* m_certificateAuthorityPEM;</td>
<td>A base64 encoded PEM format CA root certificate.</td>
</tr>
<tr>
<td>bool m_authenticateClient;</td>
<td>Ensures that both the client and server authenticate the PEM certificate. This setting is made on the server. The default is false; only the server is authenticated by default.</td>
</tr>
</tbody>
</table>
Security

You can optionally use the OpenSSL library with the TCP socket driver to support encryption of all streamed network traffic between server and clients. In Lumberyard, this security feature is currently available only for the Windows and Linux operating systems. The TCP socket driver implements these security features in the `GridMate::StreamSecureSocketDriver` class.

Set up server-side cryptographic keys and certificates as in the following example.

```cpp
GridMate::StreamSecureSocketDriver server;
GridMate::StreamSecureSocketDriver::StreamSecureSocketDriverDesc desc;
desc.m_certificatePEM = myCertPEM;
desc.m_privateKeyPEM = myPrivateKeyPEM;
const AZ::u16 port = 5555;
server.InitializeSecurity(GridMate::Driver::BSD_AF_INET, nullptr, port, 1024*64, 1024 * 64, desc);
```

The `InitializeSecurity()` method is used instead of the `Initialize()` method from the stream socket driver. You can also enable client-side certificate authentication in the secure driver description. The default is server side only.

Client-side setup is similar to the server side but does not require a private key, as in the following code example.

```cpp
GridMate::StreamSecureSocketDriver client;
GridMate::StreamSecureSocketDriver::StreamSecureSocketDriverDesc desc;
desc.m_certificateAuthorityPEM = myPublicCertPEM;
client.InitializeSecurity(GridMate::Driver::BSD_AF_INET, nullptr, 0, 1024 * 64, 1024 * 64, desc);
```

For more information about cryptographic key and security certificate options, see Enabling Encryption (p. 2069).

For more information about using the TCP stream driver, see Using the TCP Stream Driver (p. 2077).

Using Windows Registered I/O with GridMate

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Starting in Lumberyard version 1.16, GridMate carrier supports Microsoft's registered I/O (RIO) socket interface. RIO is a network traffic processing interface that provides lower latency and higher throughput to the Windows operating system kernel. Lumberyard implements RIO support in the `GridMate::RIOPlatformSocketDriver` class, which is fully contained in the carrier `SocketDriver` class.

For the Lumberyard source code, see the `SocketDriver.*` and `SecureSocketDriver.*` files in the `lumberyard_version\dev\Code\Framework\GridMate\GridMate\Carrier\` directory.

Prerequisites

The `SocketDriver` class uses the `RIOPlatformSocketDriver` class seamlessly if the following three requirements are met:

1. RIO is supported on the build computer.
2. RIO is supported at runtime on the computer that runs it.
3. RIO has been enabled by passing true to the isHighPerformance parameter in the SocketDriver class or SecureSocketDriver class.

Using the RIO Platform Socket Driver

To use the RIOPlatformSocketDriver class, initialize a custom SocketDriver or SecureSocketDriver class and assign it to the CarrierDesc::m_driver field, as shown in the following example.

Example

```cpp
class MyClass : public GridMate::SessionEventBus::Handler
{
public:
  void OnSessionDelete(GridMate::GridSession* session) override;

private:
  GridMate::SocketDriver* m_driver;
};

void MyClass::JoinSession() {
  // ...

  // Create an instance of SocketDriver that disables "full" (oversized) packets, enables cross-platform support, and
  // enables high performance socket interfaces.
  m_driver = new GridMate::SocketDriver(false, true, true);

  // Before hosting or joining a GridMate session, set the CarrierDesc::m_driver
  // property to the instance of SocketDriver.
  GridMate::SocketDesc carrierDesc;
  carrierDesc.m_driver = m_driver;

  // ...
}

// At the end of the GridMate session, delete the SocketDriver instance.
void MyClass::OnSessionDelete(GridMate::GridSession* session) {
  // ...

  delete m_driver;
  m_driver = nullptr;

  // ...
}
```

Caveats

When using the RIO platform socket driver, note the following:

- **RIO bypasses conventional socket buffers** – Because programs cannot depend on operating system buffering, your game should enable m_threadInstantResponse when it uses RIO.

- **RIO buffer sizes must be large enough to handle all buffering** – RIO requires code to preallocate fixed size buffers. This preallocation is handled in the RIOPlatformSocketDriver. The RIOPlatformSocketDriver converts the carrier descriptor (p. 2014) send and receive buffer sizes into a full packet buffer count.

- **The RIO platform driver is not compatible with the GridMate m_driverIsFullPackets setting** – GridMate “full” packets are oversized packets that are usable only for certain LAN networking...
applications. Such packets are not routable or compatible with the RIO platform driver. For this reason, set `m_driverIsFullPackets` to false when you use the RIO platform driver.

## Marshalling

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Data is written to the network using `WriteBuffer`, and data received is read using `ReadBuffer`. Each buffer specifies the endianness used.

All data marshalling, whether for a `DataSet` or RPC, is written using a specialization of the `Marshaler` type. There are a number of pre-defined marshalers for fundamental types (`int32`, `uint16`, `bool`, `float`, etc), as well as other common types like containers and bitfields.

Marshalers and read/write buffers have a close relationship. A marshaler reads or writes its data types from or to the buffer. If the type is a complex type like a class or container, then that marshaler marshals each of its fields with nested marshalers. The nested invocation of marshaler types continues until a fundamental type is written to the buffer with the endianness of the network. Additional custom marshalers can be implemented to support custom types or to perform domain-based compression. Default marshalers are implemented through template specialization.

The base `Marshaler` class in GridMate follows.

```cpp
namespace GridMate
{
    template<typename T>
    class Marshaler
    {
    public:
        void Marshal(WriteBuffer& wb, const T& value);
        void Unmarshal(T& value, ReadBuffer& rb);
    };
}
```

If a `Marshaler` instance is not specified with the data set or RPC declaration, the template specialization is used.

Implementation of the default marshaler for AZCore's `Vector3` math type can be found in `Code/Framework/GridMate/GridMate/Serialize/MathMarshal.h`:

```cpp
namespace GridMate
{
    template<>
    class Marshaler<AZ::Vector3>
    {
    public:
        typedef AZ::Vector3 DataType;
        static const AZStd::size_t MarshalSize = sizeof(float) * 3;
        void Marshal(WriteBuffer& wb, const AZ::Vector3& vec) const
        {
            Marshaler<float> marshaler;
            marshaler.Marshal(wb, vec.GetX());
            marshaler.Marshal(wb, vec.GetY());
            marshaler.Marshal(wb, vec.GetZ());
        }
    };
}
```
void Unmarshal(AZ::Vector3& vec, ReadBuffer& rb) const
{
    float x, y, z;
    Marshaler<float> marshaler;
    marshaler.Unmarshal(x, rb);
    marshaler.Unmarshal(y, rb);
    marshaler.Unmarshal(z, rb);
    vec.Set(x, y, z);
}

Markers

Notice the declaration of MarshalSize above. WriteBuffer supports the concept of markers. A marker is a placeholder that can be inserted into the buffer, so its value can be filled after additional data is written to the buffer. This is useful for prepending a length field in front of the actual data. Markers require that the data that is inserted be of fixed length, and MarshalSize is used to query this length. Therefore, marshalers that write data to the marker need to declare a valid MarshalSize.

Buffers

Write Buffers

Write buffers are backed by the following three types of allocation schemes:

Dynamic – Dynamically allocated and automatically grown

Static – Fixed size, allocated on the stack

Static In Place – Uses another buffer as its backing store

By default, the write function uses the default marshaler for the data type, but you can override the marshaler to create a custom marshaler.

There are two ways to write a type to a network buffer:

1) The following example uses the default marshaler for the type passed into Write(). In this example, the float marshaler is used.

WriteBuffer wb;
w.Write(1.0f);

2) The following example uses the HalfMarshaler, which compresses the float by half.

WriteBuffer wb;
w.Write(1.0f, HalfMarshaler());

Read Buffers

Read buffers have built-in overflow detection and do not read any data fields after the end of the buffer has been reached. You can check this by looking at the return value of the Read method. Note that if data isn’t read for a given value, then the value is left uninitialized.

Predefined Marshalers

GridMate includes the following predefined marshalers:
Fundamental C++ Types

<table>
<thead>
<tr>
<th>Floating point</th>
<th>Misc</th>
<th>Unsigned</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>char</td>
<td>AZ::u8</td>
<td>AZ::s8</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>AZ::u16</td>
<td>AZ::s16</td>
</tr>
<tr>
<td>enum (specify marshaled size by inheriting enum from a type)</td>
<td></td>
<td>AZ::u32</td>
<td>AZ::s32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AZ::u64</td>
<td>AZ::s64</td>
</tr>
</tbody>
</table>

Container Types

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Associative</th>
<th>Explicit Marshalers</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>map</td>
<td>ContainerMarshaler</td>
</tr>
<tr>
<td>list</td>
<td>set</td>
<td>MapContainerMarshaler</td>
</tr>
<tr>
<td>string</td>
<td>unordered_map</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unordered_set</td>
<td>(Use these marshalers when the subtypes of the container require a non-default marshaler)</td>
</tr>
<tr>
<td></td>
<td>multimap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>multiset</td>
<td></td>
</tr>
</tbody>
</table>

Utility Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConversionMarshaler&lt;SerializedType, OriginalType&gt;</td>
<td>Performs static casts between SerializedType (type on the wire) and OriginalType (type declared in user code).</td>
</tr>
<tr>
<td>AZ::Crc32</td>
<td>A CRC32 value.</td>
</tr>
<tr>
<td>AZStd::bitset</td>
<td>A class for arbitrary flags.</td>
</tr>
<tr>
<td>AZStd::pair</td>
<td>A std::pair class. Implicitly used by the map, unordered_map, and multimap marshalers.</td>
</tr>
<tr>
<td>AZ::Aabb</td>
<td>An axis aligned bounding box.</td>
</tr>
<tr>
<td>AZStd::chrono::duration</td>
<td>A time duration in 32 bit milliseconds.</td>
</tr>
<tr>
<td>GridMate::UnionDataSet</td>
<td>A type safe tagged union designed for network transmission.</td>
</tr>
</tbody>
</table>

Compression Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float16Marshaler</td>
<td>Compresses a float32 to float16.</td>
</tr>
</tbody>
</table>
Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HalfMarshaler</td>
<td>Compresses a float to half precision.</td>
</tr>
<tr>
<td>IntegerQuantizationMarshaler&lt;Min, Max, Bytes&gt;</td>
<td>Quantizes an integer in the range [Min, Max] to the number of bytes specified in Bytes.</td>
</tr>
</tbody>
</table>

Custom Marshalers

Creating a custom data marshaler is as simple as specializing the `GridMate::Marshaler` type, and implementing the expected `Marshal` and `Unmarshal` methods. If the data written is constant size, adding the member `MarshalSize` allows you to use the marshaler in scenarios where fixed sizes are required (such as markers).

Fixed Size Custom Marshaler

The following is an example of a fixed size custom marshaler.

```cpp
namespace GridMate
{
    template<class MyClass>
    class Marshaler
    {
    public:
        static const AZStd::size_t MarshalSize = sizeof(m_field1) + sizeof(m_field2);
        void Marshal(GridMate::WriteBuffer& wb, const MyClass& value) const
        {
            wb.Write(value.m_field1);
            wb.Write(value.m_field2);
        }
        void Unmarshal(MyClass& value, ReadBuffer& rb) const
        {
            rb.Read(value.m_field1);
            rb.Read(value.m_field2);
        }
    };
}
```

Sessions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

GridMate session service provides session connectivity and management. Both hub-and-spoke (client/server) and P2P full-mesh topologies are supported.

You can also create multiple sessions for each GridMate instance. Each session creates its own carrier and replica manager instances, so there is no interaction between sessions. GridMate sessions support host migration when running in P2P mode.

Topics

- Starting and Stopping the Session Service (p. 2026)
- Hosting a Session (p. 2028)
- Searching for a Session (p. 2029)
Starting and Stopping the Session Service

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The session service is responsible for hosting or joining sessions and is represented by the `GridMate::SessionService` abstract class.

When a session service is created, a descriptor class derived from `GridMate::SessionServiceDesc` is passed in as a constructor argument.

The implementations of `GridMate::SessionService` that are included with the base Lumberyard engine are as follows.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Descriptor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GridMate::LANSessionService</code></td>
<td><code>GridMate::SessionServiceDesc</code></td>
<td>Sessions hosted over a local area network.</td>
</tr>
</tbody>
</table>

Starting a Session Service

Only one session service can be present per `GridMate::IGridMate` instance.

**Note**
Attempting to register multiple session services causes an assert and overrides any previously registered session services.

You have two ways to start a session service:

- Create a session service object and register it with GridMate.
- Register an existing session service object with GridMate.

<table>
<thead>
<tr>
<th>Starting Method</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GridMate::StartGridMateService()</code></td>
<td>Creates a session service object and registers it with <code>GridMate::IGridMate</code>.</td>
</tr>
<tr>
<td><code>GridMate::RegisterService()</code></td>
<td>Registers an existing session service object with <code>GridMate::IGridMate</code>.</td>
</tr>
</tbody>
</table>

Stopping a Session Service

The method for stopping the session service depends on how the session service was started.

<table>
<thead>
<tr>
<th>Starting Method</th>
<th>Stopping Method</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GridMate::StartGridMateService()</code></td>
<td><code>GridMate::StopGridMateService()</code></td>
<td>The session service is stopped when <code>GridMate::IGridMate</code> is destroyed by</td>
</tr>
<tr>
<td>Starting Method</td>
<td>Stopping Method</td>
<td>Details</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>GridMate::RegisterService</td>
<td>GridMate::UnregisterService</td>
<td>The session is stopped and memory freed when GridMate::UnregisterService() is called.</td>
</tr>
<tr>
<td>GridMate::RegisterService</td>
<td>GridMate::UnregisterService</td>
<td>using the GridMate::GridMateDestroy() method.</td>
</tr>
</tbody>
</table>

**Examples**

The following examples assume that GridMate has been initialized.

**Starting and Stopping with GridMate::StartGridMateService**

The following example uses GridMate::StartGridMateService.

```cpp
void MyClass::StartSessionService()
{
    IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
    if(gridMate)
    {
        // The session service is started and will be stopped when IGridMate is destroyed.
        GridMate::SessionServiceDesc desc;
        GridMate::StartGridMateService<GridMate::LANSessionService>(gridMate, desc);
    }
}
```

**Starting and Stopping with GridMate::RegisterService() and GridMate::UnregisterService()**

The following example uses GridMate::RegisterService() and GridMate::UnregisterService().

```cpp
void MyClass::StartSessionService()
{
    IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
    GridMate::SessionService* sessionService = nullptr;
    if(gridMate)
    {
        GridMate::SessionServiceDesc desc;
        sessionService = aznew GridMate::LANSessionService(desc);
        gridMate->RegisterService(sessionService);
    }
    return sessionService;
}

void MyClass::StopSessionService(GridMate::SessionService* sessionService)
{
    IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
    if(gridMate)
    {
        // Unregister the session service and free the session service pointer.
        gridMate->UnregisterService(sessionService);
    }
}
```
Hosting a Session

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A session can be hosted by calling `IGridMate::HostSession()` after the session service has been started. The session settings and configuration are set in the `GridMate::SessionParams` argument, which acts as a base class for certain implementations of `GridMate::SessionService`.

<table>
<thead>
<tr>
<th>Implementation of GridMate::SessionService</th>
<th>Implementation of GridMate::SessionParams</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridMate::LANSessionService</td>
<td>GridMate::LANSessionParams</td>
</tr>
</tbody>
</table>

GridMate::SessionParams

The following table shows the supported parameters in `GridMate::SessionParams`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_localMember</td>
<td>Yes</td>
<td></td>
<td>This is not required for a LAN session, only for consoles.</td>
</tr>
<tr>
<td>m_topology</td>
<td>No</td>
<td>ST_PEER_TO_PEER(CLIENT_SERVER): A client is only connected to the server.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ST_PEER_TO_PEER: A client is connected to all other clients.</td>
<td></td>
</tr>
<tr>
<td>m_peerToPeerTimeout</td>
<td>Yes</td>
<td>10000</td>
<td>The time without a response, in seconds, after which a peer is disconnected.</td>
</tr>
<tr>
<td>m_numPublicSlots</td>
<td>Yes</td>
<td></td>
<td>The maximum number of players that can join the session.</td>
</tr>
</tbody>
</table>

GridMate::LANSessionParams

`GridMate::LANSessionParams` has the following additional parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_port</td>
<td>No</td>
<td>0</td>
<td>The port to monitor for search requests from other clients. If 0, this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>session is hidden to searches. Otherwise, the port number falls in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>range from 1 through 65536.</td>
</tr>
</tbody>
</table>

Events

The following table describes GridMate session service events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSessionCreated</td>
<td>A new session has just been created.</td>
</tr>
<tr>
<td>OnMemberJoined</td>
<td>A player has joined the session.</td>
</tr>
</tbody>
</table>
### Overview

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnMemberLeaving</td>
<td>A player has left the session.</td>
</tr>
</tbody>
</table>

### Examples

The following example hosts a session. The example assumes that GridMate has been initialized and a session service registered.

```cpp
bool MyClass::HostSession()
{
    GridMate::IGridMate* gridMate = gEnv->pNetwork->GetGridMate();

    if(gridMate)
    {
        GridMate::LANSessionParams params;
        params.m_topology = GridMate::ST_CLIENT_SERVER;
        params.m_numPublicSlots = 10;
        params.m_port = 10000;
        params.m_flags = 0;
        params.m_localMember = gridMate->GetOnlineService()->GetUser();

        GridMate::Session session = gridMate->HostSession(&params,
            GridMate::CarrierDesc());
        if(session != nullptr)
        {
            // Failed to create the session..
            return true;
        }
    }
    return false;
}
```

### Searching for a Session

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You search for a session by calling `GridMate::StartGridSearch()` after the session service has been started. The session settings and configuration are set in the `GridMate::SearchParams`, which acts as a base class for certain implementations of `GridMate::SessionService`.

### Implementation of `GridMate::SessionService` vs. `GridMate::SearchParams`

<table>
<thead>
<tr>
<th>Implementation of <code>GridMate::SessionService</code></th>
<th>Implementation of <code>GridMate::SearchParams</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>GridMate::LANSessionService</td>
<td>GridMate::LANSearchParams</td>
</tr>
</tbody>
</table>

### GridMate::SearchParams

The following table shows the supported parameters in `GridMate::SearchParams`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_localMember</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
GridMate::LANSearchParams

GridMate::LANSessionParams has the following additional parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_serverAddress</td>
<td>No</td>
<td>Empty</td>
<td>The address of a server to search for. If empty, a broadcast address is used.</td>
</tr>
<tr>
<td>m_serverPort</td>
<td>Yes</td>
<td></td>
<td>The port that game servers monitor for searches.</td>
</tr>
<tr>
<td>m_broadcastFrequencyMs</td>
<td>No</td>
<td>1000</td>
<td>The interval, in milliseconds, between search broadcast requests.</td>
</tr>
</tbody>
</table>

Search Results

When a search is complete, the OnGridSearchComplete() event is called. The results are found in the GridMate::GridSearch argument.

GridMate::GridSearch contains an array of search results.

To query the size of the array, use GridMate::GridSearch::NumResults().

To query individual results, use GridMate::GridSearch::GetResult().

The GridMate::SearchInfo object contains more details about the session (for example, the number of used and free player slots) and can be used when Joining a Session (p. 2031).

Events

The following table describes GridMate session search events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnGridSearchStart</td>
<td>A grid search has started.</td>
</tr>
<tr>
<td>OnGridSearchComplete</td>
<td>A grid search has finished and contains the results.</td>
</tr>
</tbody>
</table>

Examples

The following example searches for all available sessions. The example assumes that GridMate has been initialized, a session service has been registered, and the class MyClass is listening for session events.

```cpp
void MyClass::StartSearch()
{
```
GridMate::IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
if(gridMate)
{
    GridMate::LANSearchParams params;
    params.m_serverPort = 20000;
    params.m_localMember = gridMate->GetOnlineService()->GetUser();
    gridMate->StartGridSearch(&params);
}
void MyClass::OnGridSearchComplete(GridMate::GridSearch* search)
{
    if(search->GetNumResults() > 0)
    {
        // Found sessions that match the specified criteria
    }
}

## Joining a Session

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You have two ways to join a session:

- By **Searching for a Session (p. 2029)** and using a `GridMate::SearchInfo` object from the results.
- Directly to an existing game session by using a `GridMate::SessionIdInfo` object.

Regardless of the method, a session is joined using one of the overloaded `IGridMate::JoinSession()` functions after the session service has been started.

**Note**
The argument `GridMate::JoinParams` currently has no supported parameters.

### Events

The following table describes GridMate session join events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSessionJoined</td>
<td>The client has been successfully added to the session.</td>
</tr>
<tr>
<td>OnMemberJoined</td>
<td>A player has joined the session.</td>
</tr>
<tr>
<td>OnMemberLeaving</td>
<td>A player has left the session.</td>
</tr>
</tbody>
</table>

### Example

The following example joins a session that has been found as the result of a session search.

```cpp
void MyClass::OnGridSearchComplete(const GridMate::GridSearch* search)
{
    GridMate::IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
    // Further code...
}
```
if(gridMate)
{
    if(search->GetNumResults() > 0)
    {
        GridMate::Session* session = gridMate->JoinSession(search->getResult(0),
        GridMate::JoinParams(), GridMate::CarrierDesc());
    }
}
void MyClass::OnSessionJoined(GridMate::GridSession* session)
{
    // Joined the session successfully
}

Reacting to Session Events

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Much of the session functionality is asynchronous because functions can be called, but the response is often not immediately available. For example, messages may be slowed by network transfer time, server processing, or the required response time.

The Working with the Event Bus (EBus) system (p. 1851) in Lumberyard is an event bus system that can send out events when asynchronous session functions are complete. This topic shows you how to set up your application to use the event bus and to connect and disconnect from it.

Setup

Your application must derive a class from GridMate::SessionEventBus::Handler. This class must contain certain overridden session events. However, not all events need to be implemented. An example follows.

class MyClass : public GridMate::SessionEventBus::Handler
{
    public:
        void OnSessionJoined(GridMate::GridSession* session) override;
        void OnMemberJoined(GridMate::GridSession* session, GridMate::GridMember* member)
        override;
        void OnMemberLeaving(GridMate::GridSession* session, GridMate::GridMember* member)
        override;
};

Connect

The following example shows how to connect to the session event bus and start receiving session events.

void MyClass::Init()
{
    GridMate::IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
    if(gridMate)
    {
        GridMate::SessionEventBus::Handler::BusConnect(gridMate);
    }
}
**Disconnect**

The following example shows how to disconnect from the session event bus and stop receiving session events.

```cpp
void MyClass::Term()
{
    GridMate::IGridMate* gridMate = gEnv->pNetwork->GetGridMate();
    if(gridMate)
    {
        GridMate::SessionEventBus::Handler::BusDisconnect(gridMate);
    }
}
```

**Network Session Service Event Descriptions**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A description of each session event follows.

**virtual void OnSessionServiceReady()**

Callback that occurs when the session service is ready to process sessions.

**virtual void OnGridSearchStart(GridSearch* gridSearch)**

Callback when a grid search begins.

**virtual void OnGridSearchComplete(GridSearch* gridSearch)**

Callback that notifies the title when a game search query is complete.

**virtual void OnGridSearchRelease(GridSearch* gridSearch)**

Callback when a grid search is released (deleted). It is not safe to hold the grid pointer after this event.

**virtual void OnMemberJoined(GridSession* session, GridMember* member)**

Callback that notifies the title when a new member joins the game session.

**virtual void OnMemberLeaving(GridSession* session, GridMember* member)**

Callback that notifies the title that a member is leaving the game session.

**Warning**

The member pointer is not valid after the callback returns.

**virtual void OnMemberKicked(GridSession* session, GridMember* member)**

Callback that occurs when a host decides to kick a member. An OnMemberLeaving event is triggered when the actual member leaves the session.

**virtual void OnSessionCreated(GridSession* session)**

Callback that occurs when a session is created. After this callback it is safe to access session features. The host session is fully operational if client waits for the OnSessionJoined event.

**virtual void OnSessionJoined(GridSession* session)**

Called on client machines to indicate that the session has been joined successfully.
virtual void OnSessionDelete(GridSession* session)

Callback that notifies the title when a session is about to be terminated.

**Warning**
The session pointer is not valid after the callback returns.

virtual void OnSessionError(GridSession* session, const string& errorMsg)

Called when a session error occurs.

virtual void OnSessionStart(GridSession* session)

Called when the game (match) starts.

virtual void OnSessionEnd(GridSession* session)

Called when the game (match) ends.

virtual void OnMigrationStart(GridSession* session)

Called when a host migration begins.

virtual void OnMigrationElectHost(GridSession* session, GridMember*& newHost)

Called to enable the user to select a member to be the new Host.

**Note**
The value is ignored if it is null, if the value is the current host, or if the member has an invalid connection ID.

virtual void OnMigrationEnd(GridSession* session, GridMember* newHost)

Called when the host migration is complete.

virtual void OnWriteStatistics(GridSession* session, GridMember* member, StatisticsData& data)

Called at the last opportunity to write statistics data for a member in the session.

**Replicas**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Game sessions use replicas to synchronize the state of the session. To use a replica, you simply declare the states that must be synchronized and the remote procedure calls (RPCs) that are supported. After you bind the replica object to the network, the engine does the work. There is no need to worry about how to properly route messages or discover remote objects. When you add a local (primary) replica to the network, the replica is automatically discovered by remote nodes. In addition, corresponding remote proxy replica objects are created on the remote nodes. Only the owner of the replica is allowed to change states, and new states are automatically propagated to all other nodes. RPCs can be called from any node but are routed to the primary (owner) node for verification and processing.

**Topics**
- Replica (p. 2035)
- Replica Chunks (p. 2037)
- Datasets (p. 2040)
- Remote Procedure Calls (RPCs) (p. 2044)
Replica

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Replicas are core components of GridMate's replication system that are created by network-connected GridMate peers. When a peer creates a replica, GridMate propagates the replica over the network to synchronize the replica's state across the session. A locally created and owned replica is called a primary replica. The copy of the primary replica that connected peers receive is called a proxy replica. The synchronization and instantiation of replicas is handled by Replica Manager (p. 2047).

Replica Chunks

Every replica holds a collection of user-defined ReplicaChunk (p. 2037) objects that are synchronized with all the peers in the current session. A replica chunk is a container for user-defined DataSet (p. 2040) objects and Remote Procedure Calls (RPCs) (p. 2044). Any change to a DataSet object or a call to an RPC causes the replica to synchronize its state across the session.

Limitations

Replica chunks have the following limitations:

- Each replica can contain only 32 chunks.
- Chunks can be attached or detached only when a replica is not bound to a replica manager.

Creating a Replica and Attaching Chunks

To create a replica, invoke the following method:

```
GridMate::ReplicaPtr replica = GridMate::Replica::CreateReplica();
```

Most use cases require only one chunk per replica. To create a chunk and attach it to a replica by using a single call, use the CreateAndAttachReplicaChunk helper function, as in the following example:

```
GridMate::CreateAndAttachReplicaChunk<MyReplicaChunk>(replica, ...);
```

If you just want to attach a chunk to a replica, do the following:

```
replica->AttachReplicaChunk(myChunk);
```

For more information about the creation and propagation of replica chunks, see Replica Chunks (p. 2037).

Binding a Replica to the Session Replica Manager

In order for a replica to be synchronized, it must be bound to the session replica manager. After you create a replica and attach chunks to it, get the replica manager from the GridMate session (p. 2025). Then, bind the replica to it as follows:

```
GridMate::ReplicaManager* replicaManager = session->GetReplicaMgr();
replicaManager->AddMaster(replica);
```

Proxy replicas are automatically instantiated by remote peers' replica managers and, therefore, automatically bound.
Replica Ownership

When a peer creates a replica and binds it to the session replica manager, that peer becomes the owner of the replica. Each replica can be owned by only one peer. The replica owner is the only peer on the network that has the authority to change the state of the replica. For example, it can change the chunks’ datasets or directly execute its RPCs. Any state changes performed on a proxy replica are considered invalid and do not propagate throughout the session. RPCs can be called on a proxy replica, but the calls are forwarded to the owner for confirmation before they can be executed. Once this confirmation is given, the RPC is sent to all proxies and also executed locally by the peer. If the primary replica denies the execution, no peers receive the RPC call.

Changing Ownership

Replica ownership can be transferred from one peer to another, but the current owner of the replica must agree to the transfer. For information on how a replica owner can prevent transfer of ownership, see Replica Chunks (p. 2037).

Ownership transfer happens automatically when a session performs host migration on a peer-to-peer network. You can also request it explicitly by invoking the following method:

```cpp
replica->RequestChangeOwnership(); // Request ownership of a given replica for the local peer
```

Ownership transfer is an asynchronous process. When an ownership transfer is completed, each replica chunk is notified of the change by the `OnReplicaChangeOwnership` callback function.

Replica ID

Each replica has a unique ID associated with it. The replica ID is guaranteed to be unique within a particular GridMate session. You can use the replica ID to retrieve a replica from the session replica manager, as in the following example:

```cpp
GridMate::ReplicaManager* replicaManager = session->GetReplicaMgr();
GridMate::ReplicaPtr replica = replicaManager->FindReplica(myReplicaId);
if (replica == nullptr)
{
    // Replica with given ID does not exist
    return;
}
if (replica->IsProxy())
{
    // This is a proxy replica
}
if (replica->IsMaster())
{
    // This is a primary replica
}
```

Lifetime

The lifetime of a replica is controlled by a `GridMate::ReplicaPtr`, which is a reference-counted smart pointer. The replica manager retains a reference to every replica that is bound to it. It is therefore safe to omit a reference to the replica from user code; the replica is not destroyed as long as the reference is held in replica manager. However, you can force the replica manager to release its reference and free the replica by invoking the following method:
replica->Destroy();

Sample Code

This example creates a user-defined chunk, creates a replica, attaches the chunk to the replica, and binds the replica to the session replica manager.

```cpp
// User-defined ReplicaChunk class to be carried with the replica
class MyChunk : public GridMate::ReplicaChunk {
public:
    GM_CLASS_ALLOCATOR(MyChunk);
    typedef AZStd::intrusive_ptr<MyChunk> Ptr; // smartptr to hold the chunk
    static const char* GetChunkName() { return "MyChunk"; } // Unique chunk name
    bool IsReplicaMigratable() override { return false; } // Replica ownership
        // cannot be changed
    MyChunk() : m_data("Data", 0) {} // chunk constructor

    void OnReplicaActivate(const ReplicaContext& rc) override // Called when replica is bound
        // to the replica manager
    {
        // printing out whether it is a proxy or a primary replica
        if (IsMaster())
            printf("I am primary!\n");
        if (IsProxy())
            printf("I am proxy!\n");
    }

    GridMate::DataSet<int> m_data; // data this chunk holds
};
GridMate::ReplicaPtr replica = GridMate::Replica::CreateReplica(); // Creating a replica
GridMate::CreateAndAttachReplicaChunk<MyChunk>(replica); // Creating chunk of our custom type
        // and attaching it to the replica
GridMate::ReplicaManager* replicaManager = session->GetReplicaMgr(); // Getting replica manager instance
        // from current
session
replicaManager->AddMaster(replica); // Binding replica to the replica manager,
        // making local peer the owner of this replica

... // Starting from this point and up until replica destruction, the replica and MyChunk object
// that the replica is carrying are synchronized with other peers.
// Other peers receive the new replica and bind it to their replica managers. When this is done,
// OnReplicaActivate is triggered, and the "I am proxy" message is printed out on the remote peers.
// Every change of m_data DataSet results in the synchronization of the new value in
// the primary replica with all of the proxy replicas.
```

Replica Chunks

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
A replica chunk is a user extendable network object. One or more ReplicaChunk objects can be owned by a replica, which is both a container and manager for replica chunks. A replica is owned by a primary peer and is propagated to other network nodes as a proxy replica. The data that a replica chunk contains should generally be related to the other data stored within it. Since multiple chunks can be attached to a replica, unrelated data can be stored in other chunks within the same replica.

A replica chunk can contain Datasets (p. 2040) and/or Remote Procedure Calls (RPCs) (p. 2044). Datasets store arbitrary data, which only the primary replica is able to modify. Any changes are propagated to the chunks in proxy replicas on the other nodes. RPCs are methods that can be executed on remote nodes. They are first invoked on the primary, which decides whether the invocation will be propagated to the proxies.

**Replica Chunk Requirements and Limitations**

A replica chunk has several important attributes:

- It can have up to 32 DataSet definitions.
- It can have up to 256 RPC definitions.
- It is reference counted and therefore must be held by a smart pointer.
- It is not synchronized across the session until the replica manager is ready.

**Implementing a New Replica Chunk Type**

You have two ways to implement a new replica chunk type: handle data set changes and RPC calls ("game logic") inside the chunk, or outside the chunk. In both cases, the following apply:

- The name of the chunk type must be unique throughout the system. To achieve this, every replica chunk type must implement the static member function `const char* GetChunkName()`. The string returned by the `GetChunkName` function must uniquely identify the chunk type.
- To indicate whether the ownership of this type of chunk is transferrable, every chunk type needs to override the `bool IsReplicaMigratable()` virtual function. If any chunk in a replica is not migratable, the replica's ownership cannot be transferred from one peer to another.
- Every chunk type must define a smart pointer that holds the chunk type instances.

**Declaring a Replica Chunk Type with Internal Game Logic Handling**

To have your replica chunk class handle game logic directly, it should inherit from ReplicaChunk:

```cpp
class MyChunk : public GridMate::ReplicaChunk
{
public:
    GM_CLASS_ALLOCATOR(MyChunk); // Using GridMate's allocator

    MyChunk()
        : m_data("Data", 0) // Initializing integer DataSet to zero, and
        , MyRpcMethodRpc("MyRpcMethodRpc") // Initializing RPC by passing in its name; the
            // RPC name is for debugging purposes
    {
    }

typedef AZStd::intrusive_ptr<DataSetChunk> Ptr; // Defining smart pointer type
for this chunk

    static const char* GetChunkName() { return "MyChunk"; } // Unique chunk type name
    bool IsReplicaMigratable() override { return false; } // Specify whether the chunk
        // can participate in replica's ownership changes
    bool MyRpcMethod(int value, const GridMate::RpcContext& context)
}
```
Declaring a Replica Chunk Type with External Game Logic Handling

To have your replica chunk class act as a simple data carrier and forward data changes and events to a designated handler (an external class), inherit your handler class from ReplicaChunkInterface, and your replica chunk class from ReplicaChunkBase:

```cpp
class CustomHandler : public GridMate::ReplicaChunkInterface
{
public:
    GM_CLASS_ALLOCATOR(CustomHandler); // using GridMate's allocator

    void DataSetHandler(const int& value, const GridMate::TimeContext& context)
    {
        // Handle changes
    }

    bool RpcHandler(AZ::u32 value, const GridMate::RpcContext &context)
    {
        // Handle event here
        return true; // Propagate this call to all proxies
    }
};

class MyChunk : public GridMate::ReplicaChunkBase
{
public:
    GM_CLASS_ALLOCATOR(MyChunk); // Using GridMate's allocator

    MyChunk()
    : m_data("Data", 0) // Initializing integer DataSet to zero and assigning a name for it
    , MyRpcMethodRpc("MyRpcMethodRpc") // Initializing RPC by passing its name; the RPC's name is used for debugging purposes
    {
    }

    typedef AZStd::intrusive_ptr<DataSetChunk> Ptr; // Defining smart pointer type for this chunk
    static const char* GetChunkName() { return "MyChunk"; } // Unique chunk type name
    bool IsReplicaMigratable() override { return false; } // Whether chunk can participate in replica's ownership changes

    GridMate::DataSet<int>::BindInterface<CustomHandler, &CustomHandler::DataSetHandler> m_data;
    GridMate::Rpc<GridMate::RpcArg<AZ::u32>>::BindInterface<CustomHandler, &CustomHandler::RpcHandler> MyRpcMethodRpcPC;
};
```

Registering Chunk Type

Every user-defined replica chunk type should be registered with ReplicaChunkDescriptorTable to create the factory required by the Replica Manager (p. 2047).
To register replica chunks, use this call:

```cpp
GridMate::ReplicaChunkDescriptorTable::Get().RegisterChunkType<MyChunk>();
```

### Attaching a Replica Chunk to the Replica

You must add a replica chunk to a replica before you bind the replica to replica manager. After you bind the replica to replica manager, you cannot add or remove replica chunks to or from the replica.

To create a replica chunk, use this call:

```cpp
MyChunk::Ptr myChunk = GridMate::CreateReplicaChunk<MyChunk>({...});
```

Where `{...}` is forwarded to the `MyChunk` constructor.

To attach the chunk to a replica, use this call:

```cpp
replica->AttachReplicaChunk(myChunk);
```

Alternatively, you can create the chunk and attach it in one step:

```cpp
GridMate::CreateAndAttachReplicaChunk<MyChunk>(replica, {...});
```

After you add the chunk to the replica, the replica retains a smart pointer to the chunk. The chunk is released only when its replica is destroyed.

### Datasets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use `DataSet` objects to synchronize the state of a session across the network. When a value in the dataset changes, the updates are propagated automatically. Datasets can be of any type, but they must support the assignment and comparison operators. Your `DataSet` declaration can specify a custom marshaler. If you do not specify a marshaler, the `DataSet` object uses `GridMate::Marshaler<T>`.

A `DataSet` must be declared inside a `ReplicaChunk` object. A `ReplicaChunk` object can contain up to 32 `DataSet` objects. You must supply a debug name to the dataset constructor.

The following example declares a `ReplicaChunk` object that has two `DataSet` objects of type `float`. One dataset uses the default marshaler. The other dataset uses a custom marshaler called `MyCustomMarshaler`.

```cpp
class MyChunkType : public GridMate::ReplicaChunk
{
    public:
        MyChunkType()
            : m_synchedFloat("SynchedFloat")
            , m_synchedHalf("SynchedHalf")
        {
        }

        GridMate::DataSet<float> m_synchedFloat;
        GridMate::DataSet<float, MyCustomMarshaler> m_synchedHalf;
};
```
Datasets can be optionally bound to a callback on the chunk interface so that the callback is called when new data arrives.

class MyChunkType : public GridMate::ReplicaChunk
{
public:
    MyChunkType()
        : m_synchedFloat("SynchedFloat")
    {
    }

    // Callback to call when new data arrives.
    void OnSynchedFloatData(const float& newValue, const GridMate::TimeContext& timeContext);

    GridMate::DataSet<float>::BindInterface<MyChunkType, amp;MyChunkType::OnSynchedFloatData> m_synchedFloat;
};

**Eventual consistency** is guaranteed for datasets. Normally, datasets propagate unreliably. To compensate for potential packet losses, and to minimize latency, GridMate handles events in the following order:

1. A user changes a value in the dataset.
2. The new value is broadcast to the remote peers.
3. The dataset stops changing.
4. A user-configurable grace period elapses.
5. A last update is sent reliably.
6. To conserve bandwidth, propagation is suspended until the next change.

You can change the length of the grace period in step 4 by calling `SetMaxIdleTime`:

```cpp
...
GridMate::DataSet<Vector3> m_pos;
...
...
... m_pos.SetMaxIdleTime(5.f);  // Suspend sending if m_pos has not changed for 5 ticks
...```

**Carrier ACK Feedback**

Lumberyard 1.12 introduced, as a preview feature, carrier ACK feedback.

In the default GridMate implementation, a change in a dataset causes four unreliable updates and then one reliable update to be sent. This technique can result in many reliable packets on the network. Because reliable packets require ordering, preceding packets cannot be processed until a lost packet is successfully retransmitted. The resulting delay can cause jitter.

To avoid this issue, you can enable carrier ACK feedback.

When carrier ACK feedback is enabled, a changed dataset propagates its update unreliably until the receiver sends an ACK to acknowledge that it received the update. This approach removes the requirement for updates to be sent reliably (step 5 in the previous section). Additionally, if a dataset update is acknowledged as received before the grace period specified by `MaxIdleTime`, GridMate saves bandwidth by not sending additional unneeded updates.
To enable carrier ACK feedback, set the `k_enableAck` property of `ReplicaTarget` to `true` in the `ReplicaTarget.cpp` file, as in the following example:

```cpp
...  
bool ReplicaTarget::k_enableAck = true;  
...
```

### Examples

The examples in this section show three ways to create datasets.

#### Example 1

The following example creates a `DataSet` object that uses the default marshaler to store a `u32` value.

```cpp
GridMate::DataSet<AZ::u32> m_data;
```

#### Example 2

The following example creates a `DataSet` object that stores a float. The data written to the network is half float size because of the specified marshaler.

```cpp
GridMate::DataSet<float, HalfMarshaler> m_data;
```

#### Example 3

The following example creates a `DataSet` object that stores an `s32` value using the default marshaler for `s32`. Whenever the `DataSet` value changes, the `DataSetHandler` function is called on the `MyReplicaChunk` instance. This is true for both primary and proxy nodes; the event is triggered on local data changes for the primary and upon received data changes for the proxies.

```cpp
class MyReplicaChunk : public GridMate::ReplicaChunk  
{  
  void DataSetHandler(const AZ::s32& value, const GridMate::TimeContext& context) { /* Data Changed Logic */  
    GridMate::DataSet<AZ::s32>::BindInterface<MyReplicaChunk, &MyReplicaChunk::DataSetHandler> Data;  
};
```

### Throttlers

Datasets can be throttled based on an optional throttler parameter to the template. The throttler can choose to send data or withhold downstream updates unless a certain condition has been met. The throttler must implement the `WithinThreshold` method using the following syntax.

```cpp
bool WithinThreshold(T previousValue, T currentValue);
```

The return value of the method determines whether to send the data to the proxy peers.

### Using Delta Compression in Lumberyard Networking

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use Lumberyard networking's delta compression feature to reduce the amount of network traffic generated by dataset (p. 2040) updates.

By default, when a change is made to a non-compressed dataset, GridMate propagates the change by sending all of the data in the dataset across the network. Such a dataset is declared in the following way:

```cpp
GridMate::DataSet<float> m_myFloat;
```

An update to such a dataset uses four bytes on the network. To reduce this, you can use a compression marshaler like the `HalfMarshaler` located in `lumberyard_version\dev\Code\Framework\GridMate\GridMate\Serialize\CompressionMarshal.h`.

The following syntax specifies a dataset that uses the `HalfMarshaler` compression marshaler:

```cpp
GridMate::DataSet<float, HalfMarshaler> m_myFloat;
```

This compression marshaler reduces the network payload to two bytes, but at the cost of some precision.

**Delta Compression Overview**

Starting in Lumberyard version 1.18, you can use GridMate's delta compression feature to further reduce the payload to only one byte for each dataset update.

Delta compression reduces the network payload by sending only the data that changed (the "delta") instead of all data in the dataset. For example, if a float changes from 0.2 to 0.3, you can represent the change as +0.1 and encode the change in a single byte. Compared to a regular dataset, this reduces the average payload for float values by 66 percent compared to a regular dataset, or to 50 percent compared to a half marshaler compressed dataset.

GridMate supports delta compressed datasets for the two most frequently used types: `float` and `AZ::Vector3`.

The following syntax shows how to declare delta compressed datasets.

```cpp
#include <GridMate/Replica/DeltaCompressedDataSet.h>
DeltaCompressedDataSet<float, 5> m_myDeltaCompressedFloat;
DeltaCompressedDataSet<AZ::Vector3, 1> m_myDeltaCompressedVector3;
```

`DeltaCompressedDataSet` is a template that takes at least two parameters: a type and the delta range value as an unsigned integer. The delta range parameter is described in the next section.

**Delta Range**

The delta range parameter, which is unique to delta compression datasets, specifies the precision of delta compression. This value acts like a threshold. When the dataset has changed by an amount greater than the specified delta range, all data in the dataset is sent.

In the previous example, if the starting value is 0, data changes that remain within the range of -5 and +5 are compressed into one byte.

For calculation purposes, the precision of delta compression is double the delta range divided by 255, which is the size of one byte. For a delta range of 5, the precision is $5 \times 2 / 255 = 0.04$.

**Important**

Pay close attention to the required precision of your values. The highest precision for a `DeltaCompressedDataSet` is declared with a delta range of 1, as in the following example:
The precision in this case is \(1\times\frac{2}{255}=0.0078\).

**Note**
You can work around this precision limit by changing the measurement unit that you use. For example, instead of sending meters, you can send millimeters.

### Calculating Payload Reduction

Suppose a delta compressed dataset is declared in the following way:

```cpp
DeltaCompressedDataSet<float, 5> m_myDeltaCompressedFloat;
```

If a change is less than five (the configurable value specified in the second template parameter) then the update is delta-encoded in one byte. Otherwise, a full value is sent.

The following table illustrates the behavior of a sample delta compressed dataset in chronological order.

<table>
<thead>
<tr>
<th>Chronological Order</th>
<th>Value</th>
<th>Bytes Sent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>4</td>
<td>Because this is the first value, it is sent as a full update.</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>1</td>
<td>The change is less than five, so a one-byte delta is sent.</td>
</tr>
<tr>
<td>3</td>
<td>4.9</td>
<td>1</td>
<td>The change is still less than five, so a one-byte delta is sent.</td>
</tr>
<tr>
<td>4</td>
<td>5.1</td>
<td>4</td>
<td>The change is greater than five, so a full update is sent.</td>
</tr>
<tr>
<td>5</td>
<td>5.3</td>
<td>1</td>
<td>The change since the last full update is less than five (5.3-5.1=0.2), so a one-byte delta is sent.</td>
</tr>
</tbody>
</table>

In practice, a slowly changing float value averages 1.4 bytes for each update.

**AZ::Vector3 Values**

The behavior for compressed float datasets is also applicable to the delta compression of `AZ::Vector3` values. However, each coordinate value in the vector is compressed into one byte, for a minimum total of three bytes. This compares favorably to the twelve bytes required for an uncompressed `AZ::Vector3` value.

### Remote Procedure Calls (RPCs)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

RPCs allow games to send events or requests to remote nodes through replicas. They can be used to send messages to a specific node, or to route function calls to the authoritative node. For example, you can use RPCs to implement functions that change the position of an object. This ensures that changes happen only at the node that owns the object. For server-authoritative games, reliable RPCs can be used for sending frequent client input commands.

RPCs have the following characteristics:
Overview

- RPC arguments can be of any type, as long as a valid marshaler is provided.
- All RPC requests are routed to the primary replica.
- The RPC handler function in the primary replica chooses whether to propagate the RPC to proxy replicas.
- RPCs are not kept in the history, and late-joining clients might not receive RPCs requested before the client joined.

Like datasets, RPCs are declared as replica chunk members. An RPC handler function is bound to the RPC as part of the declaration. RPC requests are forwarded to the handler function along with the arguments and an RpcContext associated with the request.

The RPC handler function can perform additional checks before executing the request.

The handler for an RPC returns a Boolean value to GridMate. This value is used on the replica's primary node to determine whether the RPC is propagated to all proxies.

Remote procedure calls are always invoked first on the primary node for the replica. This is true whether the initial caller is a primary or proxy. The primary node's RPC handler decides whether the RPC should be propagated to the proxy nodes based on the return value of the RPC handler. The user returns true to mean “propagate to all replica proxies,” and false to mean “only invoke this RPC on the primary.”

RPCs have a constructor that requires a string. This is used for debugging and statistical purposes. Any debugging or network monitoring exposes the given RPC name. Using modern C++, the name can also be specified inline, as in the following example.

```cpp
Rpc<RpcArg<AZ::u32>>::BindInterface<MyClass, &MyClass::Func> Rpc = {"My RPC"};
```

Examples

The following examples show how RPCs can be used in GridMate.

Example 1

In the following example, Rpc1 is an RPC that takes a single parameter of type u32. It uses the default u32 marshaler.

```cpp
class MyReplicaChunk : public GridMate::ReplicaChunk
{
    bool Rpc1Handler(AZ::u32 val, const GridMate::RpcContext& context) { /* RPC Logic */ }
    GridMate::Rpc<GridMate::RpcArg<AZ::u32>>::BindInterface<MyReplicaChunk, &MyReplicaChunk::Rpc1Handler> Rpc1;
};
```

Example 2

In the following example, Rpc2 is an RPC that takes a single parameter of type s32. It uses IntegerQuantizationMarshaler, with a range from -100 to 100 and writes one byte to the wire.

```cpp
class MyReplicaChunk : public GridMate::ReplicaChunk
{
    bool Rpc2Handler(AZ::s32 val, const GridMate::RpcContext& context) { /* RPC Logic */ }
    GridMate::Rpc<GridMate::RpcArg<AZ::s32, GridMate::IntegerQuantizationMarshaler<-100, 100, 1>>::BindInterface<MyReplicaChunk, &MyReplicaChunk::Rpc2Handler> Rpc2;
};
```
Example 3

In the following example, Rpc3 is an RPC that takes two parameters; a u8 and a string. It uses the default marshalers for each argument.

```cpp
class MyReplicaChunk : public GridMate::ReplicaChunk
{
    bool Rpc3Handler(AZ::u8 val, const AZStd::string& str, const GridMate::RpcContext& context) { /* RPC Logic */ }
    GridMate::Rpc<GridMate::RpcArg<AZ::u8>, GridMate::RpcArg<const AZStd::string&>>::BindInterface<MyReplicaChunk, &MyReplicaChunk::Rpc3Handler> Rpc3;
};
```

Example 4

If you want to send a custom class as an RPC parameter, you must first write a marshaler for it, as in the following example.

```cpp
struct MyClass
{
    AZ::Crc32 m_name;
    AZ::u32 m_value;
};
namespace GridMate
{
    template<>
    class Marshaler<MyClass>
    {
        public:
        static const AZStd::size_t MarshalSize = Marshaler<AZ::Crc32>::MarshalSize + sizeof(AZ::u32);
        void Marshal(WriteBuffer& wb, const MyClass& value) const
        {
            wb.Write(value.m_name);
            wb.Write(value.m_value);
        }
        void Unmarshal(MyClass & value, ReadBuffer& rb) const
        {
            rb.Read(value.m_name);
            rb.Read(value.m_value);
        }
    };
}
```

An RPC that passes a parameter of the foregoing class might be declared like this:

```cpp
class MyReplicaChunk : public GridMate::ReplicaChunk
{
    bool Rpc4Handler(const MyClass& value, const GridMate::RpcContext& context) { /* RPC Logic */ }
    GridMate::Rpc<GridMate::RpcArg<const MyClass&>>::BindInterface<MyReplicaChunk, &MyReplicaChunk::Rpc4Handler> Rpc4;
};
```

For Rpc4, the first and only argument is a const reference to the MyClass object. The const MyClass& is specified to indicate that the Rpc4Handler function takes a const reference. This allows you to avoid making a copy of the object when it is passed to the handler function. Behind the scenes, GridMate stores a temporary value of MyClass, which is what the reference binds to. The temporary
referent is removed after the RPC has been called. You can also use this technique to marshal objects that are wrapped in smart pointers.

**Example 5**

In order to invoke an RPC on a given chunk instance, you can simply call the RPC, as in the following example.

```cpp
class MyReplicaChunk : public GridMate::ReplicaChunk
{
    bool Rpc5Handler(AZ::u32 val, const GridMate::RpcContext& context) { /* RPC Logic */ }
    GridMate::Rpc<GridMate::RpcArg<AZ::u32>>::BindInterface<MyReplicaChunk, &MyReplicaChunk::Rpc1Handler> Rpc5;
};

void Foo(MyChunkType* myChunkInstance)
{
    myChunkInstance->Rpc5(1);
}
```

Rpc5 is an RPC that takes a single parameter of type u32. It uses the default u32 marshaler. Calling Foo invokes the RPC on the replica chunk instance and passes in a value of 1.

**RPC Type Traits**

RPCs have an optional typetraits parameter. The following traits are expected in the traits class.

<table>
<thead>
<tr>
<th>Trait</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_isReliable</td>
<td>true</td>
<td>Uses reliable transmission to send the RPC.</td>
</tr>
<tr>
<td>s_isPostAttached</td>
<td>true</td>
<td>Forces any dirty datasets to also be sent reliably in advance. This is useful if the RPC relies on the data in the datasets to be up to date on the destination peer.</td>
</tr>
</tbody>
</table>

**Replica Manager**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The replica manager is a subsystem that is responsible for managing the synchronization of replicas. The replica manager is responsible for the following:

- Marshaling and unmarshaling the replicas in each peer
- Forwarding replicas from one peer to another
- Handling ownership changes of replicas
- Managing replica lifetimes

**Managing Replica Lifecycle**

The replica manager must do the following:
• Keep track of all replicas by holding a reference-counted pointer to every primary and proxy replica object.
• Guarantee consistency across the session by capturing and propagating the last state of every replica before a replica is destroyed.
• Guarantee that all proxies reach eventual consistency before a replica is deactivated.
• Release all GridMate references to a replica object when the object has been destroyed.

Topics
• Binding a New Primary Replica to Replica Manager (p. 2048)
• Retrieving Replicas from Replica Manager (p. 2048)
• How Replica Manager Updates Replicas (p. 2048)
• Task Manager (p. 2049)

Binding a New Primary Replica to Replica Manager

After a new primary replica is created, it must be bound to the replica manager as follows:

```cpp
GridMate::ReplicaManager* replicaManager = session->GetReplicaMgr(); // Get replica manager from the current session
replicaManager->AddMaster(myReplica1); // Bind replica to replica manager
replicaManager->AddMaster(myReplica2); // Bind replica to replica manager
```

Proxy replicas are bound to their session's replica managers automatically. Each ReplicaManager instance holds a reference to every replica that is bound to it. That changes only when the user calls Destroy() on the replica or when the ReplicaManager itself is destroyed.

Retrieving Replicas from Replica Manager

Every replica has a numeric identifier that is unique in the session. To find a replica by its ID, invoke FindReplica(<ReplicaId>), as in the following example:

```cpp
GridMate::ReplicaPtr replica = replicaManager->FindReplica(<myReplicaId>);
AZ_Assert(replica != nullptr, "Replica with id=%d not found.", <myReplicaId>);
```

How Replica Manager Updates Replicas

The GridMate session triggers the replica manager to perform replica updates on a continuous basis. These updates include the following actions:

• Unmarshaling
• Update from replica
• Update replicas
• Marshaling

Marshaling: Sending Data to Other Peers

Changes in a replica must be replicated to every remote peer in the GridMate session. To communicate a change in one of its replicas, a peer's replica manager serializes the replica object into a send buffer. It then sends the object to the network. Replica marshaling occurs in two main phases:

• Data Preparation – A premarshaling phase that, based on changes in the replica, determines which RPCs and DataSet objects to send. This phase also validates the data integrity of the objects to be sent.
• **Actual Marshaling** – The transformation of a replica object into a byte stream. The actual data that must be marshaled depends on how much new information the primary replica has relative to its corresponding remote proxy replica. For example, new proxy replicas require all information about the primary replica. This includes its datasets (p. 2040), RPCs (p. 2044), and construction metadata. Previously synchronized proxy replicas require only the information from the primary replica that is different, including any pending RPC calls.

**Unmarshaling: Receiving Data from Other Peers**

In unmarshaling, the replica manager communicates with the remote peers, receives and parses new data from them, and updates its own replicas accordingly. These updates can include accepting new peers, instantiating new proxy replicas, handling ownership changes, or destroying proxy replicas.

**Note**

For more information about marshaling, see Marshalling (p. 2022).

**Update from Replica: Updating Proxy Replicas**

A change in a custom ReplicaChunk (p. 2037) results in an UpdateFromChunk callback that causes all proxy replicas to update their state. RPCs from proxy and primary replicas are processed and invoked during this step.

**Update Replicas: Updating Primary Replicas Locally**

A change in a custom replica chunk results in an UpdateChunk callback that causes all primary replicas on a local peer to update their states.

**Task Manager**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The replica manager holds two task manager instances: one for updating and one for marshaling replicas. Updating tasks are executed within the replica manager's UpdateFromReplica step, while marshaling tasks are executed in the Marshal step. Tasks can execute other tasks while running. TaskManager::Add queues the tasks in an ordered list. TaskManager::Wait executes a task and waits until it finishes. When an event fires in the replica system, replica manager adds the corresponding task into TaskSystem.

Here are few examples of this behavior:

• A user changes a dataset's value within a replica. The change needs to be marshaled to other peers. The OnReplicaChanged event is called on ReplicaManager, and ReplicaMarshalTask is queued for execution. Because replicas must be sent in the order of their creation, the task's priority is based on the replica's creation time. The queued task is executed at the appropriate time within the Marshal step.

• A new proxy replica is unmarshaled. When this happens, OnReplicaUnmarshaled() is called and ReplicaUpdateTask is queued. This task's priority is always zero because the order of execution is not important. UpdateFromReplica is called to notify the user of the new replica's data.
The network binding API in the AZ framework provides a way for components to synchronize their states over the network.

To enable network synchronization for a component, you must do the following:

1. Derive the component from `NetBindable` and implement the network binding interfaces.
2. Implement a new replica chunk type and add the datasets and RPCs necessary to provide synchronization.

Topics

- Synchronizing an Entity with a NetBindingComponent (p. 2050)
- Binding Process on Remote Nodes (p. 2050)
- Unbinding Process (p. 2050)
- NetBindable Component Flexibility (p. 2050)
- Entity IDs (p. 2050)
- Synchronizing Animations Across a Network (p. 2051)
- Creating a NetBindable Component (p. 2055)
- Transform Component Interpolation (p. 2059)

Synchronizing an Entity with a NetBindingComponent

Because a special `NetBindingComponent` is responsible for the actual binding process, entities that need to be synchronized must have a `NetBindingComponent` added to them. When a game enters a multiplayer session, the `NetBindingComponent` collects replica chunks from the `NetBindable` instances on the entity and adds them to a `Replica` (p. 2035) primary. A special `NetBindingChunk` captures and stores spawning and other binding information for the entity. `NetBindingComponent` instances activated during a multiplayer session automatically start the binding process.

Binding Process on Remote Nodes

As replicas arrive at remote nodes, `NetBindingChunk` starts the entity spawning and binding process on the remote node. The binding process is completely asynchronous. The replicas become active first. Then an entity spawn request is queued. After the entity becomes available, its `NetBindable` components are bound to their corresponding chunks. Finally, the entity is activated.

Unbinding Process

When replicas are removed, affected `NetBindingComponent` instances start the unbinding process. By default, entities that are unbound from proxy replicas are deleted, but this doesn’t have to be always the case. A game can choose to keep all entities in place and seamlessly switch to single-player mode.

NetBindable Component Flexibility

A `NetBindingComponent` must exist for an entity to be bound to the network. This allows `NetBindable` components to be used in single-player modes without any additional runtime cost. `NetBindable` instances can also be disabled for each instance. This gives you the additional flexibility: The transform component can provide entity transform synchronization by default, but for special entities, a physics or animation component can provide more advanced synchronization.

Entity IDs

In Lumberyard, every entity has a unique ID so that it can be referenced in the game. Entity IDs are 64-bit strings generated using an algorithm that ensures uniqueness across computing devices. To reduce
binding complexity, the net binding system spawns entities to be bound to proxy replicas using the same ID as the primary.

The following diagram shows how the net binding system binds an entity to the network and spawns an entity. It does this with the same ID that it binds to a proxy replica.

---

**Synchronizing Animations Across a Network**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the AnimGraph (p. 545) component to add an animation graph and motion set to a character. To synchronize the AnimGraph component's parameters (its EMotion FX animation state) across the network, use these two components:

- Network Binding (p. 700)
- Anim Graph Net Sync (p. 545)

The Anim Graph Net Sync component, which is included with the EMotion FX Animation (p. 1124) gem, requires the Network Binding component. You can add the Anim Graph Net Sync component to any entity that has the Anim Graph component.

**Topics**

- Adding Animation Synchronization to an Entity (p. 2052)
- Running the Samples Project in a Server and Client Configuration (p. 2055)
Adding Animation Synchronization to an Entity

The following procedure shows how to add the Network Binding and Anim Graph Net Sync components to the Rin character in the Advanced_RinLocomotion Sample (p. 151) level of the Samples Project (p. 146).

To add animation synchronization to an entity

1. In the Project Configurator, do the following:
   a. Follow the steps in Choosing a Game Project to Open (p. 47) to set the SamplesProject as your default project.
   b. Follow the steps in Enabling Gems (p. 1064) to enable the Multiplayer gem for the Samples Project.

2. Close the Project Configurator, and then launch Lumberyard Editor (p. 182).
3. In Lumberyard Editor, choose File, Open Level, or press Ctrl+O.
4. In the Open a Level dialog box, under Levels, Samples, choose Advanced_RinLocomotion, and then click Open.
6. In the viewport, zoom in and select the Rin character.

7. In the Entity Inspector, ensure that Rin appears in the Name field.
8. In the Entity Inspector search box, enter anim.
The AnimGraph component for Rin has four parameters:

- movement_speed
- movement_direction
- attacking
- jumping

Changes in these parameters are replicated to other network peers after you add the Network Binding and Anim Graph Net Sync components.

9. In the Entity Inspector, choose Add Component.
10. In the Entity Inspector search box, enter Network.
11. Choose Network Binding.

12. In the Entity Inspector, click Add Component.
13. In the Entity Inspector search box, enter anim.
14. Choose Anim Graph Net Sync.
The Anim Graph Net Sync component is added to the Rin entity.

By default, the Anim Graph Net Sync component synchronizes only the animation parameters. For greater precision, you can also choose to synchronize active nodes and motion nodes, which uses more network bandwidth.

15. Choose File, Save or press Ctrl+S to save the level.
Running the Samples Project in a Server and Client Configuration

Now that you have added the Net Binding and Anim Graph Net Sync components to the Rin entity, you can test the synchronization by running server and game launcher applications.

To test the animation graph network synchronization

1. In Lumberyard Editor, choose Game, Export to Engine, or press Ctrl+E to export your level to the Samples Project game launcher.
2. In the message box that reports The level was successfully exported, click OK.
3. Open a Windows command console on the lumberyard_version\dev\Bin directory that you are using.
   • For Visual Studio 2017, the directory is lumberyard_version\dev\Bin64vc141\.
   • For Visual Studio 2019, the directory is lumberyard_version\dev\Bin64vc142\.
4. Run the following command to start the server:
   
   SamplesProjectLauncher.exe +mphost +map advanced_rinlocomotion

5. Wait until the server game is running and the Rin character renders fully.
6. Press Alt+Tab to return to the command prompt.
7. Run the following command to start and connect the client:

   SamplesProjectLauncher.exe +mpjoin

8. Wait until the client game is running and the Rin character renders fully.
9. Press Alt+Tab to return to the server application.
10. Use the following keys to control the Rin character:
    • To move forward, left, backward, and right, press the W, A, S, and D keys, respectively.
    • To look around, move the pointer.
    • To attack, press the left mouse button.
    • To jump, press the Space key.
11. Observe how, as you move the Rin character in the server application, the Rin character in the client application performs the same movements as its counterpart in the server.
12. In the server application, press the backtick key (`) to open the console, and then enter quit.
13. In the client application, press the backtick key (`) to open the console, and then enter quit.

Creating a NetBindable Component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For a Lumberyard component to share data on the network, it must include the NetBindingComponent. The NetBindingComponent creates a replica (p. 2035) for the component and can bind any replica chunk (p. 2037) that a component creates to the replica.

To enable networking on a component

1. Inherit the component from AzFramework::NetBindable:
2. Modify the `AZ_COMPONENT` definition to include `AzFramework::NetBindable`:

```cpp
AZ_COMPONENT(ShipComponent,"{D466FD68-96C9-45AF-8A89-59402B0350F7}",
    AzFramework::NetBindable);
```

3. Modify SerializeContext to include `AzFramework::NetBindable`:

```cpp
if (serialize)
{
    serializeContext->Class<ShipComponent, AzFramework::NetBindable, AZ::Component>()
        ...
}
```

4. Implement the `AzFramework::NetBindable` interfaces:

```cpp
// Called during network binding on the primary. Implementations should create and
// return a new binding.
virtual GridMate::ReplicaChunkPtr GetNetworkBinding() = 0;

// Called during network binding on proxies.
virtual void SetNetworkBinding(GridMate::ReplicaChunkPtr chunk) = 0;

// Called when network is unbound. Implementations should release their references to
// the binding.
virtual void UnbindFromNetwork() = 0;
```

**Notes**

- If the `AZ_COMPONENT` definition change is missing, the NetBindingComponent does not recognize
  the component when it checks for components to add to the replica.
- If the SerializeContext definition is missing, the primary replica still functions correctly. However,
  the proxy cannot match the IDs because it is not serialized as an `AzFramework::NetBindable`
  interface.
- Changes to these definitions require a re-export of levels for the static IDs to match correctly.

**Network Binding Function Details**

The following functions are available for working with component entities on the network.

**GetNetworkBinding**

The component uses this function to create its ReplicaChunk and initialize any state it wants to
synchronize across the session. This function is called only on the primary ComponentEntity. The
ReplicaChunk that is returned is automatically attached to the appropriate Replica.

**SetNetworkBinding**

This function passes a ReplicaChunk to the component and initializes the internal data of
the component to match that of the ReplicaChunk. This function is called only on the proxy
ComponentEntity instances that are already bound to an appropriate Replica.
UnbindFromNetwork

The UnbindFromNetwork function is called to stop the component from reacting to data updates from the network. This can happen, for example, when the primary no longer exists, has been deactivated, or has relinquished control to the local source.

Creating a Chunk

After you have enabled the NetBindable interface on the component, you must create a ReplicaChunk object that will store any state that the component wants to share.

```cpp
class ShipComponentReplicaChunk : public GridMate::ReplicaChunkBase
{
public:
    AZ_CLASS_ALLOCATOR(ShipComponentReplicaChunk, AZ::SystemAllocator, 0);

    static const char* GetChunkName() { return "ShipComponentReplicaChunk"; }

    ShipComponentReplicaChunk()
        : SetFiring("SetFireLaser")
        , m_playerEntityId("PlayerEntityId")
    {
    }

    bool IsReplicaMigratable()
    {
        return true;
    }

    GridMate::Rpc< GridMate::RpcArg<bool> >::BindInterface<ShipComponent, 
        &ShipComponent::SetFiringRPC, NetworkUtils::ShipControllerRPCTraits> SetFiring;
    GridMate::DataSet<AZ::EntityId>::BindInterface<ShipComponent, 
        &ShipComponent::OnNewNetPlayerEntityId> m_playerEntityId;
};
```

Note

You must reflect this new replica chunk's datasets and RPCs in the component's Reflect function.

```cpp
AzFramework::NetworkContext* netContext = azrtti_cast<AzFramework::NetworkContext*>(context);
if (netContext)
{
    netContext->Class<ShipComponent>()
        .Chunk<ShipComponentReplicaChunk>()
        .RPC<ShipComponentReplicaChunk, ShipComponent>("SetFireLaser",
            &ShipComponentReplicaChunk::SetFiring)
        .Field("PlayerEntityId", &ShipComponentReplicaChunk::m_playerEntityId);
}
```

In order for the component to react to a change in the DataSet object, one of the following must occur:

- The replica chunk must signal to the component when the change occurs (in the example, this is done using the BindInterface extension to DataSet).
- The component must poll the replica chunk and check the DataSet object for changes.
Example: Filling Out the AzFramework::NetBindable Interface

The examples below illustrate the use of GetNetworkBinding, SetNetworkBinding and UnbindFromNetwork.

GetNetworkBinding

In the following example, the component creates the new replica chunk and initializes the data to be networked. This function is called by the primary replica to retrieve the binding from the component.

```cpp
GridMate::ReplicaChunkPtr ShipComponent::GetNetworkBinding()
{
    ShipComponentReplicaChunk* replicaChunk = GridMate::CreateReplicaChunk<ShipComponentReplicaChunk>();
    replicaChunk->SetHandler(this);
    m_replicaChunk = replicaChunk;
    return m_replicaChunk;
}
```

SetNetworkBinding

In the following example, the component is bound to the supplied replica chunk. It also relinquishes its local state to the state specified by the replica chunk. This function is called on proxies to hand their binding over to the component.

```cpp
void ShipComponent::SetNetworkBinding(GridMate::ReplicaChunkPtr chunk)
{
    chunk->SetHandler(this);
    m_replicaChunk = chunk;
    ShipComponentReplicaChunk* shipControllerChunk = static_cast<ShipComponentReplicaChunk*>(m_replicaChunk.get());
    SetPlayerEntityIdImpl(shipControllerChunk->m_playerEntityId.Get());
}
```

UnbindFromNetwork

```cpp
void ShipComponent::UnbindFromNetwork()
{
    m_replicaChunk->SetHandler(nullptr);
    m_replicaChunk = nullptr;
}
```

Maintaining State

The last step is to create checks to make sure that any local modifications to the preferred networkable state do not overwrite the networked state. In addition, you must update the replica chunk whenever the local state changes and the component is in control of the state.

```cpp
void ShipComponent::OnNewNetPlayerEntityId(const AZ::EntityId& playerEntityId, const GridMate::TimeContext& tc)
{
    (void)tc;
    SetPlayerEntityIdImpl(playerEntityId);
}

bool ShipComponent::SetFiringRPC(bool firing, const GridMate::RpcContext& rpcContext)
{

if (AllowRPCContext(rpcContext))
{
    SetFiring(firing);
}
return false;
}

// Component implementation of to set firing
void ShipComponent::SetFiring(bool firing)
{
    m_isFiring = firing;

    if (!AzFramework::NetQuery::IsEntityAuthoritative(GetEntityId()))
    {
        // If the ship component is not authoritative, send an RPC update to the replica chunk
        ShipComponentReplicaChunk* shipChunk = static_cast<ShipComponentReplicaChunk*>(m_replicaChunk.get());
        shipChunk->SetFiring(firing);
    }
    else
    {
        if (m_isFiring)
        {
            EBUS_EVENT_ID(GetGun(), ShipGunBus, StartFire);
        }
        else
        {
            EBUS_EVENT_ID(GetGun(), ShipGunBus, StopFire);
        }
    }
}

void ShipComponent::SetPlayerEntityIdImpl(AZ::EntityId playerEntityId)
{
    AZ_Error("ShipControllerComponent", !m_playerEntityId.IsValid() || !playerEntityId.IsValid(), "Trying to rebind an already bound ship");
    if (m_playerEntityId != playerEntityId)
    {
        m_playerEntityId = playerEntityId;
        HandleShipSetup();

        if (m_replicaChunk && AzFramework::NetQuery::IsEntityAuthoritative(GetEntityId()))
        {
            // If you are authoritative over the entity and the component is replicated, update the value of the DataSet and propagate to clients
            ShipComponentReplicaChunk* shipChunk = static_cast<ShipComponentReplicaChunk*>(m_replicaChunk.get());
            shipChunk->m_playerEntityId.Set(m_playerEntityId);
        }
    }
}

**Transform Component Interpolation**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/3dengine) or visit the [AWS Game Tech blog](https://aws.amazon.com/press-releases/3dengine) to learn more.

The **Transform (p. 878)** component supports local interpolation of its position and rotation values when it is synchronized over the network.
You can use the interpolation of transform values to smooth changes in the movement and orientation of your entities when they are controlled by your server application. In Lumberyard, networking is accomplished through replicas that can either be primary replicas or proxy replicas. Primary replicas are typically server application controllers that set the values directly. Proxy replicas, typically on client applications, receive regular updates from the primary replicas. Due to changing network conditions, updates can be delayed or come at varying time intervals. Under these conditions, interpolation enables your clients to smoothly the movement of entities being controlled over the network. It does this by gradually modifying transform values until they match the last received values from the network.

In Lumberyard Editor, you can use the Entity Inspector to alter these settings in the Transform component's Network Sync section.

To have your entity synchronized across the network, your entity must have the Network Binding component, and you must enable Sync to replicas. The interpolation mode settings have no effect if your entity does not have the network binding component.

You can handle position and rotation interpolation separately.

Position interpolation refers to the smoothing of position between network updates and interruptions. This is useful if your objects change location and you notice visual jitter or sudden changes in orientation due to network conditions.

Rotation interpolation refers to the smoothing of rotation between network updates and interruptions. This is useful if your objects rotate and you notice visual jitter or sudden changes in orientation due to network conditions.

For each of these settings, you can either choose no interpolation at all, which is the default choice, or linear interpolation. For example, if your object never moves and thus has no need to smooth position over time, then you can leave Position interpolation mode set to None. If your object rotates and you notice visual jitter or sudden changes in its position under poor network conditions, you can set Rotation Interpolation to Linear. This setting can lead to smoother change in rotation over time.

Note
Scale interpolation is not supported in the Transform component.
Network Optimization of the Transform Component

The Transform component is optimized for network bandwidth. If your entity is synchronized over the network but only changes position, then only new position values are sent across the network. This avoids the network cost of sending the entire transform. The same is true for rotation or scale of the transform. This is done automatically. You do not have to change any settings to use this feature.

For information on providing your own interpolation logic for the Transform component, see Providing Your Own Interpolation Logic for the Transform Component (p. 2061).

Providing Your Own Interpolation Logic for the Transform Component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add your own interpolation logic for the Transform component by using a common framework available in C++. This topic shows you how to use C++, serialization, and the Lumberyard Editor UI to write your own interpolation mode.

Topics
- Adding Your Interpolation Mode to the Lumberyard Editor UI (p. 2061)
- Implementing Interpolation Logic in C++ (p. 2063)
- Rotation Interpolation (p. 2067)

Adding Your Interpolation Mode to the Lumberyard Editor UI

You can start from the user interface in the editor for the Transform component — specifically, with the interpolation options. The following image shows the Transform component's None or Linear interpolation options in the Entity Inspector.

Seeing how these options are implemented in the code can help you understand the changes that are required to add your own options.
The **None** and **Linear** interpolation options in the Entity Inspector come from the serialization of 
`AzToolsFramework::TransformComponent::Reflect`, which is the editor variation of 
`TransformComponent`.

The following related source code can be found in the file `dev\Code\Framework\AzToolsFramework\AzToolsFramework\ToolsComponents\TransformComponent.cpp`.

```cpp
namespace AzToolsFramework
{
    void TransformComponent::Reflect(AZ::ReflectContext* context)
    {
        // Reflect data for script, serialization, editing.
        if (AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
        {
            ... 
            if (AZ::EditContext* ptrEdit = serializeContext->GetEditContext())
                
            ptrEdit->Class<TransformComponent>("Transform", "Controls the placement
            of the entity in the world in 3d")
                ... 
            
            ClassElement(AZ::Edit::ClassElements::Group, "Network Sync")
                ->
            
            EnumAttribute(AzFramework::InterpolationMode::NoInterpolation, 
                "None")
                ->
            
            EnumAttribute(AzFramework::InterpolationMode::LinearInterpolation, 
                "Linear");
                ...
        }
        ...
    }
}
```

A line of code like the following adds these options to Lumberyard Editor. It contains an enum value and 
a string name for the value that appears in the editor. The reflection code for the position and rotation 
options is similar.

```cpp
EnumAttribute(AzFramework::InterpolationMode::NoInterpolation,"None")
```

`AzFramework:: InterpolationMode:: NoInterpolation` is an enum value from the 
`AzFramework` namespace. You can find the related source code in the file `dev\Code\Framework \AzFramework\AzFramework\Math\InterpolationSample.h`.

```cpp
namespace AzFramework
```
/**
 * Behavior types for smoothing of transform between network updates.
 */
enum class InterpolationMode : AZ::u32
{
    NoInterpolation,
    LinearInterpolation,
    MyInterpolation,       // <--- NEW CONTENT
};

Following this example, you can add your own enum value. The following example calls it MyInterpolation.

namespace AzFramework
{
    /**
     * Behavior types for smoothing of transform between network updates.
     */
    enum class InterpolationMode : AZ::u32
    {
        NoInterpolation,
        LinearInterpolation,
        MyInterpolation,       // <--- NEW CONTENT
    };
}

Next, update AzToolsFramework:: TransformComponent ::Reflect as in the following example.

namespace AzToolsFramework
{
    void TransformComponent::Reflect(AZ::ReflectContext* context)
    {
        // Reflect data for script, serialization, editing.
        if (AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
        {
            ...
            DataElement(AZ::Edit::UIHandlers::ComboBox,
                        &TransformComponent::m_interpolatePosition,
                        "Position Interpolation", "Enable local interpolation of position.")->
                EnumAttribute(AzFramework::InterpolationMode::NoInterpolation,
                    "None")->
                EnumAttribute(AzFramework::InterpolationMode::LinearInterpolation,
                    "Linear")->
                EnumAttribute(AzFramework::InterpolationMode::MyInterpolation, "My Mode")->       // <--- NEW CONTENT
            ...
        }
    }
}

Implementing Interpolation Logic in C++

Now that Lumberyard Editor has your new option, you can implement it in your code.

The AzToolsFramework:: TransformComponent is simply a component that the editor uses to create the game component counterpart AzFramework:: TransformComponent. Examine the
following source code for the TransformComponent class. The code shows where the interpolation samples are kept and the interpolated value is calculated. The code is in the file dev\Code\Framework\AzFramework\AzFramework\Components\TransformComponent.h.

```cpp
namespace AzFramework
{
    class TransformComponent
        : public AZ::Component,
        public AZ::TransformBus::Handler,
        public AZ::TransformNotificationBus::Handler,
        public AZ::EntityBus::Handler,
        public AZ::TickBus::Handler,
        private AZ::TransformHierarchyInformationBus::Handler,
        public NetBindable
    {
        ...

        private:
        ...

        void CreateTranslationSample();
        void CreateRotationSample();

        AZStd::unique_ptr<Sample<AZ::Vector3>> m_netTargetTranslation; // <--- Sample>

        AZStd::unique_ptr<Sample<AZ::Quaternion>> m_netTargetRotation;
        AZ::Vector3 m_netTargetScale;
        ...
    } // namespace AZ
}
```

Sample<> is the base abstract class for all interpolation logic. Notice that it's abstracted away to support either vectors for position or quaternions for rotation. The following code shows the source for the Sample class in the file dev\Code\Framework\AzFramework\AzFramework\AzFramework\Math\InterpolationSample.h.

```cpp
namespace AzFramework
{
    template<typename Value>
    class Sample
    {
    public:
        virtual ~Sample() = default;
        using TimeType = unsigned int;

        Sample()
            : m_targetValue()
            , m_targetTimestamp(0)
            , m_previousValue()
            , m_previousTimestamp(0)
        {
        }

        void SetNewTarget(Value newValue, TimeType timestamp) // <---- Network stack provides you these values every time it gets them.
        {
            m_targetValue = newValue;
            m_targetTimestamp = timestamp;
        }

        virtual Value GetInterpolatedValue(TimeType time) = 0; // <---- Provide your own interpolation logic here.
    }
}
```
The simplest implementation is no interpolation at all, for which the code is already written. You can find it in the same file: `dev\Code\Framework\AzFramework\AzFramework\Math\InterpolationSample.h`.

```cpp
template<typename T>
class UninterpolatedSample;

template<>
class UninterpolatedSample<AZ::Vector3> final : public Sample<AZ::Vector3>
{
public:
    AZ::Vector3 GetInterpolatedValue(TimeType /*time*/) override final
    {
        return GetTargetValue();
    }
};
```

The following code shows an implementation of linear interpolation that is included with Lumberyard (`dev\Code\Framework\AzFramework\AzFramework\Math\InterpolationSample.h`).

```cpp
namespace AzFramework
{
    template<typename T>
    class LinearlyInterpolatedSample;

    template<>
    class LinearlyInterpolatedSample<AZ::Vector3> final : public Sample<AZ::Vector3>
    {
    public:
        AZ::Vector3 GetInterpolatedValue(TimeType time) override final
        {
            AZ::Vector3 interpolatedValue = m_previousValue;
            if (m_targetTimestamp != 0)
            {
                if (m_targetTimestamp <= m_previousTimestamp || m_targetTimestamp <= time)
                {
                    if (m_targetTimestamp <= m_previousTimestamp || m_targetTimestamp <= time)
                    {
```

Version 1.28
2065
interpolatedValue = m_targetValue;
}
else if (time > m_previousTimestamp)
{
    float t = float(time - m_previousTimestamp) / float(m_targetTimestamp -
        m_previousTimestamp);
    
    // lerp translation
    AZ::Vector3 deltaPos = t * (m_targetValue - m_previousValue);
    interpolatedValue = m_previousValue + deltaPos;

    AZ_Assert(interpolatedValue.IsFinite(), "interpolatedValue is not
finite!");
}

SetPreviousValue(interpolatedValue, time);
return interpolatedValue;
};

The following example shows a completed implementation called `MyInterpolatedSample` that provides only vector interpolation for position.

```cpp
namespace AzFramework
{
    template<typename T>
    class MyInterpolatedSample;

    template<>
    class MyInterpolatedSample<AZ::Vector3> final : public Sample<AZ::Vector3>
    {
        ...
    }
}
```

Finally, you must enable `TransformComponent` to choose your class implementation. You can do that in the switch case statement in `AzFramework::TransformComponent::CreateTranslationSample()`. The source code is in the file `dev\Code\Framework\AzFramework \AzFramework\Components\TransformComponent.cpp`.

```cpp
void AzFramework::TransformComponent::CreateTranslationSample()
{
    switch(m_interpolatePosition)
    {
        case InterpolationMode::LinearInterpolation:
            m_netTargetTranslation = AZStd::make_unique<LinearlyInterpolatedSample<AZ::Vector3>>();
            break;
        case InterpolationMode::NoInterpolation:
            default:
                m_netTargetTranslation = AZStd::make_unique<UninterpolatedSample<AZ::Vector3>>();
                break;
    }
}
```

Provide a new case statement for your enum option. The option creates an instance of your `MyInterpolationSample` class.

```cpp
void AzFramework::TransformComponent::CreateTranslationSample()
```
Synchronizing Game State Using Scripts

That's it! Now you can select your new option in the Entity Inspector, and your custom interpolation logic does the work.

Rotation Interpolation

The preceding example shows how to provide your own position interpolation. Rotation interpolation is similar, but with the following minor differences:

- You have to provide template implementation of `MyInterpolationSample<Quaternion>`.
- You have to write similar code in `AzFramework::TransformComponent::CreateRotationSample()` instead of in `::CreateTranslationSample()`.
- In the location for the interpolation options, the `AzToolsFramework::TransformComponent` has a separate definition for rotation interpolation.

Synchronizing Game State Using Scripts

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can synchronize game state by using the Lua `ScriptComponent`. The initial steps of synchronizing game state using the Lua `ScriptComponent` are similar to any other component. There are two main steps:

1. You must add a `NetBindingComponent` to the definition of the entity that contains the script and the `ScriptComponent` and whose state you want to synchronize.
2. Inside the script, any properties that need to be synchronized must be tagged accordingly. For more information, see Network Binding Properties (p. 2691) in the Writing Lua Scripts (p. 2683) topic.

When these steps are completed, your game state data should synchronize correctly.

Using Encryption

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
GridMate uses the OpenSSL implementation of Datagram Transport Layer Security (DTLS) to support encryption of all UDP traffic sent between clients and servers.

**Limitations**

GridMate's implementation of encryption has the following limitations:

- Only 64-bit Windows is supported.
- Only client-server topologies are supported.

**Implementation Support**

GridMate supports encryption for the following implementations:

- Server and client authentication
- Self-signed certificates
- A single strong OpenSSL cipher

**Cipher**

GridMate uses the following single OpenSSL cipher for all encrypted connections: **ECDHE-RSA-AES256-GCM-SHA384**.

This cipher uses the technologies listed in the following table:

**Cipher Technologies in GridMate**

<table>
<thead>
<tr>
<th>Technology</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDHE</td>
<td>Key exchange</td>
<td><strong>Ephemeral Elliptic Curve Diffie-Hellman</strong> anonymous key agreement protocol</td>
</tr>
<tr>
<td>RSA</td>
<td>Peer authentication</td>
<td><strong>RSA</strong> algorithm used to authenticate client and server</td>
</tr>
<tr>
<td>AES256</td>
<td>Symmetric encryption cipher</td>
<td><strong>Advanced Encryption Standard</strong> that uses a 256-bit key</td>
</tr>
<tr>
<td>GCM</td>
<td>Block cipher mode of operation</td>
<td><strong>Galois/Counter Mode</strong> authenticated encryption algorithm</td>
</tr>
<tr>
<td>SHA384</td>
<td>Hashing algorithm</td>
<td><strong>SHA-2</strong> with a 384-bit digest size</td>
</tr>
</tbody>
</table>

**Topics**

- Building with Encryption (p. 2068)
- Enabling Encryption (p. 2069)

**Building with Encryption**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
When you include the GridMate library in your project, encryption support is automatically provided. However, because the GridMate library is statically linked, you must first make some modifications to the WAF build script (p. 125) (wscript) that uses GridMate.

Building Your Project with Encryption

To use encryption with GridMate, you must modify your .wscript file to add a dependency on GridMate, link the OpenSSL library, and specify OpenSSL library paths.

To modify your .wscript file to use OpenSSL with GridMate

1. Add the following line to create a dependency on GridMate:

   ```
   use = ['GridMate']
   ```

2. Add the following line to link the OpenSSL library:

   ```
   win_lib = ['ssleay32', 'libeay32']
   ```

3. Add the OpenSSL library paths, as in the following example. Within the Lumberyard install directory, these paths are in the folder `dev\Code\SDKs\OpenSSL\lib`:

   ```
   win_x64_debug_libpath = [ bld.Path('Code/SDKs/OpenSSL/lib/vc120_x64_debug') ],
   win_x64_profile_libpath = [ bld.Path('Code/SDKs/OpenSSL/lib/vc120_x64_release') ],
   win_x64_release_libpath = [ bld.Path('Code/SDKs/OpenSSL/lib/vc120_x64_release') ],
   win_x64_debug_dedicated_libpath = [ bld.Path('Code/SDKs/OpenSSL/lib/vc120_x64_debug') ],
   win_x64_profile_dedicated_libpath = [ bld.Path('Code/SDKs/OpenSSL/lib/vc120_x64_release') ],
   win_x64_release_dedicated_libpath = [ bld.Path('Code/SDKs/OpenSSL/lib/vc120_x64_release') ]
   ```

Building Without Encryption

If your project uses GridMate, but does not require support for encryption, ensure that the GridMateForTools line is in your .wscript file:

```
use = ['GridMateForTools']
```

Enabling Encryption

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To enable encryption with OpenSSL in a GridMate session, perform the following steps.

To enable encryption in a GridMate session

1. To set the encryption parameters, create an instance of SecureSocketDesc. The parameters are described in SecureSocketDesc (p. 2070).

2. Create an instance of SecureSocketDriver that passes in the instance of SecureSocketDesc. The instance of SecureSocketDesc must be available for the duration of the GridMate session.

3. Before hosting or joining a GridMate session, define CarrierDesc by setting the CarrierDesc::m_driver property to the instance of SecureSocketDriver. If no instance
of SecureSocketDriver is provided, an unencrypted driver is used that provides plaintext communication.

4. You can delete the SecureSocketDriver instance at the end of the GridMate session, ideally in the OnSessionDelete event on the SessionEventBus.

The GridMate Session Encryption Example (p. 2071) at the end of this topic has sample code for these steps.

**SecureSocketDesc**

The constructor for SecureSocketDriver requires a SecureSocketDesc object that provides all encryption configuration required for the secure connection. The configuration parameters are described in the following table.

**SecureSocketDesc Configuration Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_privateKeyPEM</td>
<td>Base-64 encoded string PEM private key.</td>
</tr>
<tr>
<td>m_certificatePEM</td>
<td>Base-64 encoded string PEM public certificate.</td>
</tr>
<tr>
<td>m_certificateAuthorityPEM</td>
<td>Base-64 encoded string PEM certificate authority.</td>
</tr>
<tr>
<td>m_authenticateClient</td>
<td>If set to 1, the client is expected to provide a signed certificate for authentication. To implement this, m_certificatePEM must be set on the client, and the server needs to set up m_certificateAuthorityPEM. The default setting is 0.</td>
</tr>
</tbody>
</table>

**Server Authentication Only**

You can use the server authentication only configuration when the client needs to verify the authenticity of the server to which it connects. The server has a secret private key and a public certificate signed by a certificate authority. This is the most common configuration.

**Server Authentication Only Configuration**

<table>
<thead>
<tr>
<th>Role</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>m_certificateAuthorityPEM</td>
</tr>
<tr>
<td>Server</td>
<td>m_privateKeyPEM, m_certificatePEM, m_certificateAuthorityPEM</td>
</tr>
</tbody>
</table>

**Client and Server Authentication**

Use this configuration when the client must verify authenticity of the server and the server must verify authenticity of the client. The client has its own unique private key and corresponding signed public certificate. The server has its own unique private key and corresponding signed public certificate.

It's possible to share or use the same certificate authority for both, but keys and certificates must be unique to each peer.

**Client and Server Authentication Configuration**

<table>
<thead>
<tr>
<th>Role</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>m_privateKeyPEM, m_certificatePEM, m_certificateAuthorityPEM</td>
</tr>
</tbody>
</table>
Self-signed Certificates

You can use self-signed certificates for development purposes.

**Warning**
Do not use self-signed certificates for production environments.

When you use self-signed certificates, there is no certificate authority to sign the public certificates. To permit the absence of a certificate authority, set `m_certificateAuthorityPEM` to the same value as `m_certificatePEM`.

GridMate Session Encryption Example

The following code snippet enables encryption in a GridMate session.

```cpp
class MyClass : public GridMate::SessionEventBus::Handler
{
public:
    void OnSessionDelete(GridMate::GridSession* session) override;

private:
    GridMate::SecureSocketDriver* m_secureDriver;
};
void MyClass::JoinSession() {
    // ...
    // Create an instance of SecureSocketDesc and set its encryption parameters.
    GridMate::SecureSocketDesc secureDesc;
    secureDesc.m_privateKeyPEM = "...
    secureDesc.m_certificatePEM = "...
    secureDesc.m_certificateAuthorityPEM = "...

    // Create an instance of SecureSocketDriver that passes in the instance of SecureSocketDesc.
    m_secureDriver = new GridMate::SecureSocketDriver(secureDesc);

    // Before hosting or joining a GridMate session, set the CarrierDesc::m_driver property to the instance of SecureSocketDriver.
    carrierDesc.m_driver = m_secureDriver;

    // ...
}

// At the end of the GridMate session, delete the SecureSocketDriver instance in the OnSessionDelete event.
void MyClass::OnSessionDelete(GridMate::GridSession* session) {
    // ...
    delete m_secureDriver;
    m_secureDriver = nullptr;
    // ...
}
```
How To Generate a Private Key and Public Certificate

You can use the `openssl req` command to generate a self-signed certificate from OpenSSL, as in the following example.

```
dev/Code/SDKs/OpenSSL/bin/openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

The arguments are as follows.

- `-x509` – The certificate format.
- `-newkey` – The type of key. This example generates an RSA key with 2048 bits.
- `-keyout` – The name of the key PEM file that will be generated
- `-out` – The name of the cert PEM file that will be generated

Upon execution, the command prompts for additional user input required to generate the certificate.

Controlling Bandwidth Usage

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

GridMate (p. 1990) provides several ways to control the bandwidth that your game uses, including bandwidth throttling and the prioritization of replica (p. 2035) updates.

Controlling the Send Rate

You can use GridMate to control the server send rate, which is a common technique for reducing bandwidth usage in multiplayer games. In this scenario, a multiplayer game is hosted by a dedicated server to which clients send their replica changes at their default rate (for example, 60 frames per second). To reduce bandwidth usage, you lower the server send rate (for example, to 20 transmissions per second). To avoid jitter when this technique is used, the client must be able to interpolate the game state that it receives from the server.

To control the server send rate in GridMate, set the time interval for replica data transmissions:

```
ReplicaMgr* replicaManager = session->GetReplicaMgr(); // Get the replica manager instance.
This assumes the session has been established.
replicaManager->SetSendTimeInterval(100); // Set the send interval to 100 milliseconds. 10 updates per second will be sent.
```

Setting the `SetSendTimeInterval` to 0 sends the data at the engine's frame rate. The default is 0.

Bandwidth Limiter

Another technique is to limit outgoing bandwidth in exchange for increased latency in the replication of objects. In GridMate, you can do this by setting a bandwidth limit on replica manager. To do so, specify a byte limit for `SetSendLimit`, as in the following example:

```
ReplicaMgr* replicaManager = session->GetReplicaMgr(); // Get the replica manager instance.
This assumes the session has been established.
```
Controlling Bandwidth Usage

```cpp
replicaManager->SetSendLimit(10000);  // Set the transmission limit to 10 kilobytes per second.
```

Setting `SetSendLimit` to 0 disables the bandwidth limiter. The default is 0.

**Controlling Burst Length**

You can use the GridMate limiter to accommodate short bursts in bandwidth if your bandwidth usage is not already at its maximum. This can be useful in many game applications. For example, when a user is in a multiplayer lobby, the corresponding bandwidth usage is quite low. However, when the user joins the game, the bandwidth usage spikes as the initial game state replicates from the server to the client. To control the length of the burst permitted, specify the desired number of seconds for `SetSendLimitBurstRange`, as in the following example:

```cpp
ReplicaMgr* replicaManager = session->GetReplicaMgr();  // Get the replica manager instance.
replicaManager->SetSendLimitBurstRange(5.f);  // Set the maximum permitted length of the burst to 5 seconds.
```

Bursts in bandwidth usage are allowed for the number of seconds specified, after which the bandwidth is capped to the value set by `SetSendLimit`. The default value for `SetSendLimitBurstRange` is 10 seconds. If bandwidth usage has already reached its limit when the burst occurs, bandwidth usage continues to be capped, and the `SetSendLimitBurstRange` setting has no effect.

**Prioritization of Replica Updates**

Every replica chunk (p. 2037) has a priority that you can assign. The priority is represented by an integer from 0 through 65534. Larger integers represent higher priorities. Replicas with higher priorities are sent first. The default is 32768.

This prioritization is especially important when you use the bandwidth limiter because you can use it to define which objects are more important and which are less important. If your game has a bandwidth cap and you have prioritized your replicas appropriately, the objects with higher priority are sent more often. The objects with lower priority are sent only when there is enough bandwidth to accommodate them.

For convenience, GridMate provides five predefined priorities that you can use for custom replica chunks:

```cpp
static const ReplicaPriority k_replicaPriorityHighest = 0xFFFE;  // Decimal 65534, highest priority.
static const ReplicaPriority k_replicaPriorityHigh = 0xC000;  // Decimal 49152, high priority.
static const ReplicaPriority k_replicaPriorityNormal = 0x8000;  // Decimal 32768, normal priority. This is the Default.
static const ReplicaPriority k_replicaPriorityLow = 0x4000;  // Decimal 16384, low priority.
static const ReplicaPriority k_replicaPriorityLowest = 0x0000;  // Decimal 0, lowest possible priority.
```

By default, all chunks have normal priority (`k_replicaPriorityNormal`). You can use these predefined priorities as is, or use them to create your own, as in the following example:

```cpp
// A replica chunk with this priority will be sent before all the chunks with Normal priority, but after all the chunks with High priority:
```
const ReplicaPriority k_myCustomPriority = (k_replicaPriorityNormal + k_replicaPriorityHigh) / 2; // (=Decimal 40960)

The priority for the whole replica is the maximum priority found in its chunks. Priority for a chunk can be set after the chunk is created, or at any point during its lifetime, as in the following example:

MyChunk::Ptr myChunk = GridMate::CreateReplicaChunk<MyChunk>(...);
myChunk->SetPriority(k_replicaPriorityLow); // Sets low priority for myChunk.

Chunks with the same priority are sent and received in the order of their creation. Replicas created earlier are sent and received first.

**Tuning Bandwidth at Runtime**

You can tune bandwidth usage while the game is running by using the following configuration variables (CVars):

<table>
<thead>
<tr>
<th>CVar</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gm_replicasSendTime</td>
<td>The time, in milliseconds, between replica transmissions. A value of 0 binds the interval to the GridMate tick rate.</td>
</tr>
<tr>
<td>gm_replicasSendLimit</td>
<td>The limit, in bytes, of the amount of replica data that can be sent per second. A value of 0 disables the limit.</td>
</tr>
<tr>
<td>gm_burstTimeLimit</td>
<td>The time, in seconds, that bursts in bandwidth are allowed. Bursts are allowed only if the bandwidth is not capped when the burst occurs.</td>
</tr>
</tbody>
</table>

**Using Bit Packing in Lumberyard Networking**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Most built-in C++ types use more than one byte in memory. Even a Boolean value uses one byte. On some operating systems, it can use more. However, the Lumberyard networking system, GridMate, optimizes networking traffic by tightly packing your data into the network payload.

**Boolean Values**

GridMate is intelligent enough to pack a Boolean value into a single bit. For example, if a replica chunk contains several Boolean fields, each field uses a single bit during network transmission. In the following example, GridMate serializes the payload for MyChunk into just three bits.

```cpp
class MyChunk : public ReplicaChunk
{
    ...
    public:
        ....DataSet<bool> m_field1; // This field occupies only one bit.
        ....DataSet<bool> m_field2; // This field occupies only one bit.
        ....DataSet<bool> m_field3; // This field occupies only one bit.
};
```
Tip

Using AZStd::bitset is unnecessary and is often less efficient. Because AZStd::bitset writes its payload in full bytes, packing one Boolean or eight Booleans into AZStd::bitset takes a whole byte. It is better to use DataSet<bool> instead of DataSet<AZStd::bitset>.

Implementing Bit Packing for a Custom Class

The following steps show you how to implement bit packing for a custom class.

1. Declare an Integer Variable That Uses Only the Required Number of Bits

If you have some custom types that you want to pack efficiently, declare an integer variable that uses only the required number of bits. The following simple example declares a flags variable for storing flags in a bit field.

```cpp
struct CustomClass
{
    int flags : 4; // C declaration of an integer value that uses only 4 bits.
};
```

2. Provide a Custom Marshaler

For best results, provide a custom marshaler as in the following example:

```cpp
class MarshalerCustomClass
{
    public:
    void Marshal(WriteBuffer& wb, const CustomClass& value) const
    {
        AZ::u8 tmp = value.flags;
        wb.WriteRaw(&tmp, { 0, 4 }); // Writes 4 bits and 0 bytes from 'tmp'.
    }

    void Unmarshal(CustomClass& value, ReadBuffer& rb) const
    {
        AZ::u8 tmp;
        rb.ReadRaw(&tmp, { 0, 4 }); // Reads just 4 bits and no full bytes.
        value.flags = tmp;
    }
};
```

3. Pass the Marshaler Type as a DataSet Argument

Now, when you declare a DataSet for a CustomClass variable, you can simply pass the marshaler type into the template arguments for DataSet. The following example shows the syntax.

```cpp
DataSet<CustomClass, MarshalerCustomClass> m_value;
```

In this implementation, the example uses a total of only 4 bits to serialize CustomClass.

4. Read and Write Data at the Bit Level

The following example shows how ReadBuffer supports direct control over reading data at the bit level.

```cpp
bool ReadBuffer::ReadRaw(void* source, PackedSize size);
// PackedSize is a special type that you can use to define granularity at the bit level.
// For example PackedSize(0, 1) means 1 bit.
// PackedSize(4, 5) means 4 bytes and 5 bits.
```
The `WriteBuffer` method has the same capability. The following example uses both the `ReadBuffer` and `WriteBuffer` methods.

```cpp
/*
 * This example takes advantage of the bit packing feature in GridMate.
 */
struct CustomClass3
{
    int flags : 7;  // Uses only seven bits.
    bool b;         // Uses only one bit.
};
/*
 * Marshaler for PackedSize objects.
 */
class MarshalerCustomClass3
{
public:
    void Marshal(WriteBuffer& wb, const CustomClass3& value) const
    {
        AZ::u8 tmp = value.flags;
        wb.WriteRaw(&tmp, { 0, 7 });
        wb.WriteRawBit(value.b);
    }
    void Unmarshal(CustomClass3& value, ReadBuffer& rb) const
    {
        AZ::u8 tmp;
        rb.ReadRaw(&tmp, { 0, 7 });
        value.flags = tmp;
        rb.ReadRawBit(value.b);
    }
};
 DataSet<CustomClass3, MarshalerCustomClass3> m_field;
```

**Setting up a Lobby**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

By default, the Lumberyard engine does not provide any specific lobby implementation, but instead provides the code interface required to construct one. The Multiplayer Gem (p. 1171) does, however, provide some useful constructs that aid in lobby creation using a basic lobby implementation using components (p. 532), that can be used as is, or as a reference.

For more information, see the Multiplayer Gem (p. 1171).

**Creating Dedicated Servers**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
In Lumberyard, you can create a build of your game that runs on a headless dedicated server. A
dedicated server build omits the packages and modules that a headless server does not require and has a
smaller compiled size.

Resources to Include

The package that you build for the dedicated server must include the resources that are required to run
the game on the client. You can use Lumberyard’s asset pipeline to exclude certain resources that are not
needed on the server. For more information, see Configuring the Asset Pipeline (p. 269).

Client and Server

Amazon GameLift splits sessions between client and server, but Lumberyard does not. Lumberyard
recommends that you not compile critical server logic on clients or include client data in the server
package.

Waf Build Configuration

To generate a dedicated server build that you can use locally or upload to Amazon GameLift, attach
the _dedicated label to your build configuration in Waf. For more information on using Waf to set
up the dedicated server, see Creating a Dedicated Server (p. 158). The topic is part of the Multiplayer
Sample (p. 155), which includes an example of a dedicated server.

Linux Support

Lumberyard has a Linux version of the dedicated server. For information on how to compile and build
dedicated server packages in Linux, see Creating Lumberyard Executables for Linux (p. 3269).

Using the TCP Stream Driver

To use the TCP stream socket driver, you specify connection parameters for internal buffers, a listening
port, and the maximum number of connections.

After you construct the new stream driver, you can use the socket driver for server or client work. For
server work, the driver can monitor for connections. For client work, the driver can connect to a server’s
endpoint address and port.

To start accepting new connections on the server, call the StartListen() method. When new clients
connect or disconnect, the socket driver sends events through the event bus.

During the setup for the carrier instance, call the StartListen() method for servers and the
ConnectTo() method for clients. You can do this before you assign the m_driver member of the
CarrierDesc structure.

Stream Connection Setup

To construct a stream socket driver, specify the maximum number of connections (1 for clients), the
maximum packet size, and the byte sizes for the internal buffers for inbound and outbound traffic.

To prepare the socket driver, use the Initialize() method with a socket family. IPv4 and IPv6 are
supported. Specify binding parameters and socket buffer sizes. The driver uses internal buffers to spool
streamed traffic during the `Update()` call. GridMate processes the `Update()` call when it is using the carrier instance.

**Constructing the TCP Socket Driver**

The following code example shows how to construct the TCP socket driver.

```cpp
// Specify the maximum number of connections that the server accepts.
const AZ::u32 maxConnections = 32;

// Specify the largest packet to be sent.
const AZ::u32 maxPacketSize = 1024 * 64;

// Specify the size of the internal byte buffer that stores incoming bytes from the socket.
const AZ::u32 inboundBufferSize = 1024 * 64;

// Specify the size of the internal byte buffer that stores outbound bytes to the socket.
const AZ::u32 outboundBufferSize = 1024 * 64;

GridMate::StreamSocketDriver server(maxConnections, maxPacketSize, inboundBufferSize, outboundBufferSize);
```

Note the following:

- The sizes are specified in bytes.
- The maximum packet size should be less than or equal to the outbound buffer size.
- Both buffer sizes are stored in application memory.

**Initializing the Socket Driver**

The following code example shows how to initialize the TCP socket driver.

```cpp
// For IPv4, specify driver BSD_AF_INET. For IPv6, specify BSD_AF_INET6.
const AZ::s32 familyType = GridMate::Driver::BSD_AF_INET;

// Specify the address of the Ethernet card to bind to.
const char* address = "127.0.0.1";

// Specify the port for the server to monitor.
const AZ::u32 serverPort = 2017;

// Specify that TCP sockets cannot send broadcast packets.
const bool isBroadcast = false;

// Set the receive buffer size for each new socket.
const AZ::u32 receiveBufferSize = 1024 * 256;

// Set the send buffer size for each new socket.
const AZ::u32 sendBufferSize = 1024 * 64;
server.Initialize(familyType, address, serverPort, isBroadcast, receiveBufferSize, sendBufferSize);
```

Note the following:

- Both server and client have to use the same socket family `AF_*` type.
- The address can be `nullptr`, which means bind to all the available network interfaces.
- The `isBroadcast` flag is always set to `false`, since a TCP socket is not able to send broadcast packets.
- The receive and send buffer size are specified in bytes and refer to the TCP stack's buffer size for the socket.
Setting Up a Stream Socket Event Bus Handler

An event bus handler instance monitors connection events from GridMate’s stream socket driver. The following code shows a simple stream socket event handler.

```cpp
struct ConnectionHandler : public GridMate::StreamSocketDriverEventsBus::Handler
{
    ConnectionHandler(GridMate::IGridMate* gridMate)
    {
        m_ConnectionCount = 0;
        GridMate::StreamSocketDriverEventsBus::Handler::BusConnect(gridMate);
    }
    ~ConnectionHandler()
    {
        m_ConnectionCount = -1;
        GridMate::StreamSocketDriverEventsBus::Handler::BusDisconnect();
    }
    void OnConnectionEstablished(const SocketDriverAddress& address) override
    {
        m_ConnectionCount++;
    }
    void OnConnectionDisconnected(const SocketDriverAddress& address) override
    {
        m_ConnectionCount--;
    }
    bool DidConnect()
    {
        return m_ConnectionCount > 0;
    }
    AZ::s32 m_ConnectionCount;
};
```

Processing a Server

After you construct and initialize the stream socket driver, use the `StartListen()` method to instruct the server to accept clients. The `StartListen()` method takes a `backlog` parameter that specifies the incoming connection request limit. The driver emits successful connections to its event bus.

When you are finished with the server process, but before you shut down GridMate, call the `StopListen()` method. To disconnect a client, you can use the `DisconnectFrom()` method on the server’s driver instance.

Processing a Client

To connect a client to a server, create a socket driver address. To start the connection process, use the `ConnectTo()` method. When the stream socket driver establishes a connection to the server, it sends a connect event.

To disconnect from the server, the client code uses the `DisconnectFrom()` method. The driver sends a disconnect event when the server gracefully disconnects from the client.

Sending and Receiving Data from the Stream Driver

To send and receive data from the stream driver, you use the `Send()` and `Receive()` methods just as you do with the UDP driver.

For both sending and receiving, the `data` and `data size` values should be equal to or greater than the maximum packet size. If the `Send()` method does not return `GridMate::Driver::EC_OK`, then the connection to the remote endpoint has been closed.
The `Receive()` method returns the number of bytes written to the data buffer. If it returns zero (0) bytes and returns a result code of `GridMate::Driver::EC_OK`, then no more packets are ready to be read.

### Using GridMate for Large-Scale Worlds

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

GridMate is Lumberyard's networking system. Interest manager is Lumberyard's solution to large-scale networked worlds.

**Topics**

- The Relationship Between Replica Manager and Interest Manager (p. 2080)
- Interest Manager (p. 2081)
- Slice Caching and Entity Lifecycle Management (p. 2082)
- Multiplayer Sample Implementation (p. 2082)
- Proximity Net Interest Component (p. 2084)
- Writing Your Own Interest Manager Attributes, Rules and Rule Handlers in C++ (p. 2086)

### The Relationship Between Replica Manager and Interest Manager

Interest manager is an optional feature that controls the sending of replicas in large-scale network game applications. Replica Manager (p. 2047) works without interest manager. However, replica manager sends replicas to all peers. Thus, all associated entities with the Network Binding component on them appear on all peers. When the Interest Manager component is created and initialized, it tells the replica manager to no longer broadcast all replicas to all peers. Interest manager acts as an overseer of replica manager and coordinates replicas and peers.
Broadcasting all replicas to all peers is not feasible for large-scale networked worlds. For these applications, you can use interest manager to control which replicas are broadcast to which peers and under what conditions. The following diagram shows the relationship between interest manager, clients, and replicas.

### Interest Manager Update Logic

1. **Init**
   - SetAutoBroadcast to false

2. **Loop** (for each rule handler)
   - **Update**

3. **Loop** (for each rule handler)
   - **GetLastResult**
   - **InterestMatchResult**
     - process matches
   - **OnReplicaChanged**
     - if a replica list of peers have changed
     - Marshal
     - removes or adds replicas from peers as necessary

---

**Interest Manager**

Broadcasting all replicas to all peers is not feasible for large-scale networked worlds. For these applications, you can use interest manager to control which replicas are broadcast to which peers and under what conditions. The following diagram shows the relationship between interest manager, clients, and replicas.
In this arrangement, replicas have attributes. Each peer can have rules that define the conditions under which attributes are matched and sent to or removed from that peer. Your server or authoritative peer can have several rule handlers that match rules to attributes. Interest manager does the work of matching and merging results and sending or removing replicas from the appropriate peers.

Suppose that you want to build a spatial system that replicates objects based on where your client peer is located within your world. An attribute would be a geometrical volume that represents the location and bounds of the replica. A rule would define a geometrical volume that your client considers in its vicinity of interest. A rule handler would perform smart spatial matching among these spatial objects.

At a low level, GridMate uses the following constructs to implement attributes, rules, and handlers:

```cpp
GridMate::ProximityInterestAttribute
GridMate::ProximityInterestRule
GridMate::ProximityInterestHandler
```

The Multiplayer Sample (p. 155) that is included with Lumberyard contains a practical implementation of these. For detailed information on writing your own attributes, rules, and rule handlers, see Writing Your Own Interest Manager Attributes, Rules and Rule Handlers in C++ (p. 2086).

### Slice Caching and Entity Lifecycle Management

NetBindingSystem caches slices for interest manager. Slice instances are cached and reused for the entities from the same slice instance. Whenever a replica is activated, the cache is searched to see if a slice that contains the entity is already in the cache. If the lookup is successful, the entity is reused.

When a game entity is no longer needed (that is, UnBindGameEntity is called), NetBindingSystem decides whether to deactivate the entity or destroy it. If the entity is in the cache, NetBindingSystem deactivates the entity. If the entity is not found in the cache, NetBindingSystem destroys the entity.

### Multiplayer Sample Implementation

The Multiplayer sample shows how to achieve interest-based filtering of networked entities in your own application.

Because the goal is to control entities, interest manager must work with entities and components. The Multiplayer sample uses the following components to accomplish this:

- Interest Manager
- Game Player Net Interest
- Proximity Net Interest

While the sample is not comprehensive, it provides a starting point for you to understand the technology and implement a system that matches your needs.

### Interest Manager Component

Lumberyard’s AzFramework::InterestManagerComponent is intended for use as a system component in your application.

The InterestManagerComponent initializes both the interest manager and the following built-in rule handlers:

- Proximity rule handler (GridMate::ProximityInterestHandler) – Use the proximity rule handler to specify, based on their spatial proximity, which entities appear in each of your peers. The rule handler uses axis-aligned bounding boxes.
• Bitmask rule handler (GridMate::BitmaskInterestHandler) – Use the bitmask rule handler to filter out replicas based on their bit mask value. This is useful for entities whose presence is determined by custom grouping.

To register InterestManagerComponent as a system component, override AZ::Module::GetRequiredSystemComponents(). As a system component, the InterestManagerComponent is initialized before any other nonsystem entities and nonsystem components. For more information, see System Components (p. 1098).

For reference, see the following code. The source file is \dev\MultiplayerSample\Gem\Code\Source\MultiplayerSampleModule.cpp.

class MultiplayerSample
    : public CryHooksModule
{
    public:
        MultiplayerSample()
            : CryHooksModule()
        {
            // System Components
            RegisterSystemComponent<AzFramework::InterestManagerComponent>(); // enabling interest management
            ...
        }
    template<class T>
        void RegisterSystemComponent()
        {
            m_descriptors.push_back(T::CreateDescriptor());
            m_systemComponents.push_back(azrtti_typeid<T>())
        }
    /**
     * Add required SystemComponents to the SystemEntity.
     */
    AZ::ComponentTypeList GetRequiredSystemComponents() const
    {
        return m_systemComponents; // this is how Lumberyard will get your system components and attach them on your behalf
    }
    ...
    AZ::ComponentTypeList m_systemComponents;
};

Game Player Net Interest

The game player net interest rule (MultiplayerSample::GamePlayerNetInterest) defines the entities in which the game player is interested and contains the GridMate::ProximityInterestRule. The console variable mps_interestRadius determines the radius of the game player's interest. The Multiplayer sample uses the following procedure to create the player entity. You can find the source code in the file dev\MultiplayerSample\Gem\Code\Source\Components\Spawn\PlayerSpawnComponent.cpp.

void PlayerSpawnComponent::SpawnPlayerEntity()
{
    if (m_playerEntity == nullptr)
    {
        m_playerEntity = aznew AZ::Entity("Game Player");
        if (m_playerEntity)
        {
            m_playerEntity->CreateComponent<GamePlayerComponent>();
        }
    }
}
m_playerEntity->CreateComponent<GamePlayerNetInterest>();
m_playerEntity->CreateComponent<AzFramework::NetBindingComponent>();
m_playerEntity->Init();
m_playerEntity->Activate();
AZ::EntityBus::MultiHandler::BusConnect(m_playerEntity->GetId());
}
...
}

The rule is created in GamePlayerNetInterest::BindTargetEntity(), which is called when the replica of this entity is bound. The source code location is \dev\MultiplayerSample\Gem\Code \Source\Components\Networking\GamePlayerNetInterest.cpp.

void GamePlayerNetInterest::BindTargetEntity(const AZ::EntityId& entityId)
{
    ... PeerId peerId = InvalidReplicaPeerId;
    EBUS_EVENT_ID_RESULT(peerId, GetEntityId(), GamePlayerInterfaceBus, GetPeerId); // Find our own peer ID.
    ProximityInterestHandler* proximityInterest = nullptr;
    EBUS_EVENT_RESULT(proximityInterest, AzFramework::InterestManagerRequestsBus, GetProximityInterest);
    if (proximityInterest)
    {
        m_proximityRule = proximityInterest->CreateRule(peerId); // Finally, create the interest manager rule.
    }
    ...}

After the rule is created, the spatial information can be set at any time. The Multiplayer sample uses OnTick to update the player's location. GamePlayerNetInterest retrieves the latest transform of the player location and sets the local proximity rule accordingly. The source code is in GamePlayerNetInterest.cpp.

void GamePlayerNetInterest::OnTick(float deltaTime, AZ::ScriptTimePoint time)
{
    ... AZ::Transform worldTM;
    EBUS_EVENT_ID_RESULT(worldTM, m_targetEntityId, AZ::TransformBus, GetWorldTM); // This updates your local proximity rules that define which entities you are interested in.
    m_proximityRule->Set(AZ::Aabb::CreateCenterRadius(worldTM.GetPosition(), m_interestRadius));
}

**Proximity Net Interest Component**

The Proximity Net Interest component describes the attribute for the entity to which it is attached. The component listens to the transform changes of the entity and updates its internal attribute accordingly. This allows interest manager to control the entity's presence on peers based on the peers' game player net interest. In the **Entity Inspector**, the name of this component is **Proximity Interest attribute**.
Entities that are filtered by interest manager have the Proximity Net Interest component attached. This component listens to transform changes of TransformComponent and updates its GridMate attribute accordingly. The following is the related code snippet from `dev\MultiplayerSample\Gem\Code\Source\Components\Networking\ProximityNetInterest.cpp`.

```c++
void ProximityNetInterestComponent::OnTransformChanged(const AZ::Transform& localTM, const AZ::Transform& worldTM)
{
    ...
    AZ::Aabb bbox = AZ::Aabb::CreateNull();
    CollisionInfo info;
    EBUS_EVENT_ID_RESULT(info, GetEntityId(), CollidableBus, GetCollisionInfo); // Get our bounding box.
    if (info.m_shape)
    {
        bbox = info.m_shape->GetEncompassingAabb();
    }
    m_attribute->Set(bbox); // Update GridMate::ProximityInterestAttribute value for the interest manager.
    ...
}
```
Writing Your Own Interest Manager Attributes, Rules and Rule Handlers in C++

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can freely extend the GridMate interest manager system with your own logic. To create a custom rule handler, inherit and implement the `GridMate::BaseRulesHandler` C++ interface. The following snippet shows the code for the `BaseRulesHandler` class.

```cpp
// dev\Code\Framework\GridMate\GridMate\Replica\Interest\RulesHandler.h
namespace GridMate
{
    class InterestManager;

    /**
     * BaseRulesHandler: base handler class.
     * RulesHandler's job is to provide InterestManager with matching pairs of attributes and rules.
     */
    class BaseRulesHandler
    {
        public:
            BaseRulesHandler() : m_slot(0) {}

            virtual ~BaseRulesHandler() { };

            /**
             * Ticked by InterestManager to retrieve new matches or mismatches of interests.
             */
            virtual void Update() = 0;

            /**
             * Returns result of a previous update.
             * This only returns changes that happened on the previous tick, not the whole world state.
             */
            virtual const InterestMatchResult& GetLastResult() = 0;

            /**
             * Called by InterestManager when the given handler instance is registered.
             */
            virtual void OnRulesHandlerRegistered(InterestManager* manager) = 0;

            /**
             * Called by InterestManager when the given handler is unregistered.
             */
            virtual void OnRulesHandlerUnregistered(InterestManager* manager) = 0;

            /**
             * Returns the InterestManager that this handler is bound to, or nullptr if it's unbound.
             */
            virtual InterestManager* GetManager() = 0;

        private:
            friend class InterestManager;
    }
```
Of interest is the method `GetLastResult`, which returns `InterestMatchResult`. `InterestMatchResult` is essentially an unordered map between replicas and a list of the peers on which the replicas should be present. If a peer is not on the list for a replica and has the replica's proxy, the interest manager removes the replica's proxy. The following code shows the declaration of the `InterestMatchResult` class.

```cpp
using InterestPeerSet = unordered_set<PeerId>;

/**
 * InterestMatchResult: a structure to gather new matches from handlers.
 * Passed to handler within matching context when handler's Match method is invoked.
 * User must fill the structure with changes that handler recalculated.
 * Specifically, the changes should have all the replicas that had their list of
 * associated peers modified.
 * Each entry replica - new full list of associated peers.
 */
class InterestMatchResult : public unordered_map<ReplicaId, InterestPeerSet>
```

It's up to you to decide how you want your rule handler to match up rules and attributes. To help you get started, GridMate provides some simple base classes for rules and attributes, as the following code shows.

```cpp
/**
 * Base class for interest rules.
 */
class InterestRule
{
public:
    explicit InterestRule(PeerId peerId, RuleNetworkId netId)
        : m_peerId(peerId), m_netId(netId)
    {}
    PeerId GetPeerId() const { return m_peerId; }
    RuleNetworkId GetNetworkId() const { return m_netId; }

protected:
    PeerId m_peerId; ///< the peer this rule is bound to
    RuleNetworkId m_netId; ///< network id
};

/**
 * Base class for interest attributes.
 */
class InterestAttribute
{
public:
    explicit InterestAttribute(ReplicaId replicaId)
        : m_replicaId(replicaId)
    {}
```
Using Amazon GameLift

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard supports hosting dedicated servers on the cloud by using Amazon GameLift. GameLift is a managed AWS service for deploying, operating, and scaling session-based multiplayer games. GameLift is built on AWS's highly available cloud infrastructure and allows you to quickly scale high-performance game servers up and down to meet player demand – without any additional engineering effort or upfront costs. It reduces the time required to build a multiplayer backend from thousands of hours to just minutes.

To use GameLift in your project, there are two options:

- Enable the GameLift Gem (p. 1130) in your project. Lumberyard has integrated Amazon GameLift, which makes it easier for you to use GameLift.
- Enable the Lumberyard Multiplayer Gem (p. 1171) in your project (which requires the GameLift Gem).

For information about configuring GameLift for the multiplayer sample, see Configuring the Multiplayer Sample for Amazon GameLift (p. 2003). For information, see Add modular features and assets with Gems (p. 1064). For more information about GameLift, see Amazon GameLift.

Additional Links

- Amazon GameLift - Creating game sessions and connecting (video)
- Amazon GameLift Developer Guide
- Amazon GameLift API Reference

Testing your Amazon GameLift Integration

Before you make your first Amazon GameLift fleet, verify your integration using GameLift Local. GameLift Local is included as part of the GameLift Server SDK download. It is run as a Java process.

With GameLift Local, you can test most aspects of your GameLift integration, with the exception of matchmaking. For details and other limitations, see the complete documentation for Testing Your Integration in the Amazon GameLift Developer Guide.
The following example demonstrates how to test MultiPlayerSample on Windows.

1. **Start GameLift Local.**

   Open a command prompt window, navigate to the directory containing the file GameLiftLocal.jar, and run it. By default, GameLift Local listens for requests from game clients on port 8080. To specify a different port number, use the `-p` parameter, as shown in the following example:

   ```java
   java -jar GameLiftLocal.jar -p 9080
   ```

   **Important**
   Use the same port in the `gamelift_endpoint` cvar for the game client launcher.

   When GameLift Local starts, you see logs indicating that two local websockets were started, one listening for your game server and one listening for your game client or the AWS CLI. Logs continue to report activity on the two local servers, including communication to and from your game components.

2. **Start the MultiPlayerSample dedicated server.**

   Open a new command window, navigate to the directory containing the game server, and run it using the following command-line options:

   ```
   MultiplayerSampleLauncher_Server.exe +sv_port 33435 +gamelift_start_server +map MultiplayerSample
   ```

   In the GameLift Local console, log messages indicate that your game server has connected to the GameLift Local service, initialized the GameLift Server SDK with `InitSDK()`, and called `ProcessReady()`. If successful, you are now ready to host a game session.

   **Tip**
   For details on these and other calls, see GameLift interactions.

3. **Start the MultiPlayerSample client.**

   Start the launcher using the following command-line options:

   ```
   MultiplayerSampleLauncher.exe +sv_port 33435 +gamelift_fleet_id fleet-123 +gamelift_uselocalserver 1 +gamelift_endpoint 127.0.0.1:9080 +gamelift_aws_access_key any_string +gamelift_aws_secret_key any_string +gamelift_region us-west-2
   ```

   **Note**
   Note that the `gamelift_uselocalserver` cvar is set to 1 to indicate the use of a local server, and the `gamelift_endpoint` cvar is set to the local endpoint with the same port used to start GameLift Local. You do not need real AWS credentials when testing local; you can set the access key and secret key to any string.

**Troubleshooting Amazon GameLift integration**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following information to help you diagnose and fix common problems that you might encounter while working with Amazon GameLift.

**Topics**
Fleets failing to go active

As a first step, sign in to the Amazon GameLift console and check the fleet's events to get more details on the problem. Here you can examine the output of the install script and get more information regarding where GameLift is seeing problems.

Missing dependencies

If your process is failing on launch, then the most likely cause is that you failed to package or install all of the server's dependencies. On Windows, this typically means missing dynamic link libraries (DLLs). Check the following:

- Check your `install.bat` or `install.sh` output. Ensure that the script installed what was required to run on GameLift.

  **Note**
  For Windows fleets, debug builds of the dedicated server will fail on launch, since they require debug Visual Studio Redistributables which Lumberyard cannot provide. Consider using profile builds instead on GameLift.

  **Note**
  For Linux fleets, look at the `MultiplayerSample_CreateGameLiftPackage.sh` script to see how to generate an `install.sh` for use on Amazon Linux 2.

- Remote into the fleet and ensure that the `bootstrap.cfg` is correct. Many failures can be caused by a misconfigured bootstrap. Try manually running the process to discover what might be missing on launch.

- If you are using Windows to prepare packages for Linux development, ensure that the box marked **Setup for Linux Dedicated Server** is selected in the Setup Assistant, otherwise GameLift integration files can be missing. This is a requirement for using `MultiplayerSample_LinuxPacker.bat`, for example.

Crashing process

If your process is crashing, then remote into the fleet and check the server logs to discover the source of the crash. Lumberyard logs to either `C:\Game\user\log` (Windows) or `/local/game/user/log` (Linux).

Not responding to GameLift callbacks

Ensure that your process is responding to callbacks from GameLift, especially the health check callback. If your process is not responding, then ensure your Amazon EC2 instance is not CPU-bound or memory-bound, which might cause the server to be unstable or slow to respond. Try reducing the number of concurrent servers per instance, or try a larger EC2 instance.

Clients unable to connect

If your clients fail to connect to the server, then check the following configurations:

- Ensure that the expected server fleet port(s) are open in Amazon GameLift. This can be checked from the fleet console and the CLI.

- Ensure that your server is listening on the port reported to GameLift via the `ProcessParameters` object passed to `ProcessReady()`.
• Ensure that your clients do not have the sv_port cvar set to something unexpected. We recommend that you leave this at the ephemeral port.

• If you are using MultiplayerSample as a basis for your fleets, verify that the server and client have the same self-signed certificate. You must use a consistent certificate across clients and server fleets. For more information about self-signed certificates, see About Self-Signed Certificates in the Multiplayer Sample (p. 158).

• Verify that gm_netsec_enable is consistent between your clients and fleets. It is on by default for the dedicated server, but you can turn it off by passing +gm_netsec_enable 0 in the startup params.

**Additional information**

For additional troubleshooting tips and information, see the following topics:

• Debug GameLift fleet issues
• Accessing GameLift fleet instances remotely
• Testing your GameLift integration
• Multiplayer Sample troubleshooting (p. 159)

**Useful Console Commands**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following commands in Lumberyard when working with a network server.

`gm_debugdraw debug_draw_level`

Sets the debug draw level. Accepts as a parameter a number who's bits represent the flags for the debug data to draw. For example, when set to 1, displays an overlay with GridMate network statistics and information.

The available bit flags come from the enum DebugDrawBits and are as follows:

```c
enum DebugDrawBits
{
    Basic           = BIT(0),
    Trace           = BIT(1),
    Stats           = BIT(2),
    Replicas        = BIT(3),
    Actors          = BIT(4),
    EntityDetail    = BIT(5),

    Full            = Basic | Trace | Stats | Replicas | Actors,
    All             = 0xffffffff,
};
```

`gm_disconnectDetection`

When set to 0, disables disconnect detection. This is useful when you are debugging a server or client and don’t want to be disconnected when stepping through code. The default value is 1.

`gm_dumpstats`

Write GridMate network profiling stats to file.
Using Multiplayer Analytics

Lumberyard User Guide Lumberyard User Guide

Using Multiplayer Analytics

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes the Multiplayer Analytics Gem for analyzing your network traffic live using an in-game overlay. When this GridMate Gem is enabled, you can bring up an in-game UI by pressing the HOME key.

gm_dumpstats_file

The file to which GridMate profiling stats are written. The default is net_profile.log.

gm_net_simulator

Activate GridMate network simulator to simulate latency, packet loss, bandwidth restrictions, and other conditions. For available options, enter gm_net_simulator help.

gm_setdebugdraw

Display an overlay with detailed GridMate networking statistics and information. A user-friendly helper command for gm_debugdraw debug_draw_level. Possible parameters are Basic, Trace, Stats, Replicas, and Actors.

gm_stats_interval_msec

Set the interval, in milliseconds, for gathering network profiling statistics. The default is 1000.

gm_tracelevel trace_level

Set the GridMate debugging trace verbosity level. The default is 0. The higher the value, the greater the verbosity. Typical values range from 1 to 3.

mpstart [<local_port>]

Starts a LAN session by initializing the network system and optionally setting the local UDP port that initializes the socket. The default port is 64090. To use the ephemeral port, set the port to 0. This is useful if you want to connect to a server on the same computer as the client.

mphost

Create a session as host. The server listens for incoming connections on the port specified in mpstart.

mpjoin [<server_addr>] [<server_port>]

Connect to a server at the optionally specified <server_addr> and <server_port>. The defaults are localhost and 64090, respectively.

map <map_name>

Loads the level with the specified map name. Replace <map_name> with the name of the map you want to use. To view a list of available levels, enter map, and then press the tab key.

mpdisconnect

Terminate the current game instance session.

sv_port local_port

Sets the local UDP port that initializes the socket. The default port is 30090. To use the ephemeral port, set the port to 0. This is useful if you want to connect to a server on the same computer as the client.
If you don't have any clients connected to your server, or you are running a single player game, you will see the following minimal UI:

Once you have a client connected to your server, you will see total bandwidth data:

In order to get detailed information of network traffic, you need to select Analyze network traffic.
Now you can expand either incoming or outgoing replica updates.

The **MultiplayerSample** level shows the network payload for each unique replica name. If you have multiple replicas that share the name, their data gets combined.

You can see the total average kilobits per second, average bytes per replica update, maximum byte per replica type update, and the overall bytes sent for a given replica name. You can also drill in deeper into each replica.
At the next level of detail you can see the network costs per replica chunks. You can drill in one more time to see the cost per DataSet and Remote Procedure Call.

### Tip
You can filter by replica name using the Filter field. For example, you can choose to only show replicas that have "small" in their name. (Case is ignored.)

Using this analytics tool, you can figure out which replicas are your most expensive network data, and then drill down to find which particular chunks and fields are the most expensive items for you. Since this tool is live and in-game, you might find it useful to record a video of your project with the overlay on while performing a particular game test, and then observe what changes occur in your network usage pattern.

### Note
All values update once every second.

---

## Engaging Broadcasters and Viewers on Twitch

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/game-tech/) or visit the AWS Game Tech blog to learn more.

Lumberyard is integrated with Twitch so that you can build games that engage with broadcasters and viewers on Twitch.

### Twitch ChatPlay

The Twitch ChatPlay feature within Lumberyard helps you build gameplay that interacts in real time with Twitch viewers. For example, you can build a game where viewers can vote on game outcomes, gift power-ups to their favorite players, or change the level based on the number of viewers watching the player.

### Twitch JoinIn

The Twitch JoinIn feature within Lumberyard helps you build multiplayer games that allow Twitch broadcasters to invite fans to join them side by side in the game. Once invited, a fan can jump into the broadcaster's game with a single click in the Twitch chat channel, while others continue to watch.

### Twitch API
Twitch API functionality in Lumberyard is based on the publicly available RESTful functions described in the Twitch API reference. Each of these functions has a corresponding call in the Lumberyard TwitchApiRequestBus that will queue an HTTP request. You can listen for the response to this request on the TwitchApiNotifyBus.

Prerequisites

To use the Twitch gem and add Twitch support to your Lumberyard project, you must:

- Be authorized as a Twitch development partner. To register, visit the Twitch Developer Portal at https://dev.twitch.tv/.

Topics

- Twitch ChatPlay System (p. 2096)
- Twitch C++ API Reference (p. 2099)

Twitch ChatPlay System

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Twitch ChatPlay provides a flexible framework to create customized game interactions between broadcasters and spectators on Twitch, the world’s leading social video platform and community for gamers.

Twitch ChatPlay includes support for chat commands, polls, and surveys that can be triggered by Twitch viewers through the Twitch chat channel. For example, you can create a chat command #cheer that triggers celebration animations in your game.

Twitch ChatPlay is implemented by a set of Script Canvas nodes that establish a connection to a Twitch channel and use incoming traffic as a game input, like any other input device.

Twitch ChatPlay includes the following components and services:

- Twitch IRC servers
- Twitch ID authentication
- Twitch account

The following diagram illustrates Twitch ChatPlay’s server-side components.
The following diagram illustrates Twitch ChatPlay's client-side components.

Using the Twitch Chat API Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Adding the Twitch Chat API Gem to your project enables you to connect to the Twitch IRC and interact with players using Twitch ChatPlay.
Twitch ChatPlay includes support for chat commands, polls, and surveys that can be triggered by Twitch viewers through the Twitch chat channel.

For gem setup steps, read the Twitch Chat API Gem (p. 1208) documentation.

Gem Usage

Once the gem has been added to your project and you have setup a Twitch account and OAuth token, you are ready to connect to Twitch IRC using C++, Lua, or Script Canvas and join a channel.

Connect to Twitch ChatPlay

Use the `ConnectToChatplay` function to connect to ChatPlay by providing your OAuth token (1st argument), and your username (2nd argument). Once connected, you should see a message confirming that Twitch is connected through the `TwitchChatPlayNotificationBus`. You will need to listen to this bus in C++ or Lua, or by simply using the Script Canvas node to receive the message.

Join a channel

Use the `JoinChannel` function to join a channel on Twitch. If your account has moderator control of that channel, you can use moderator features with your ChatPlay connection.

Game usage

Examples of how to use a ChatPlay connection in your game include the following:

- React to new subscribers of the channel.
- Listen to messages in the channel and filter them using a keyword system (p. 2098).
- Respond to specific keywords.
- Implement voting using keywords.

All of these can be implemented using a bus call in C++, Lua, or Script Canvas nodes. Look in the directory Gems\TwitchChatPlay\Assets for Lua and Script Canvas usage examples.

Keyword System

ChatPlay enables you to search incoming messages from Twitch for keywords. Your game can respond to these messages. Here are some rules and recommendations to follow when creating keywords for your game:

To create keywords

Three functions are used to set keywords in the system. The first two are also accessible through Script Canvas. Be sure to set a keyword only once, using one of the following handlers:

1. `SetKeywordWithDefaultHandler` – Adds a keyword for use throughout a session. The keyword will automatically have its handler assigned to the `TwitchChatPlayNotificationBus`, and messages will be sent through that system when the keyword is triggered.
2. `SetKeywordWithFormattedHandler` – Adds a keyword that will be parsed into username and message, in separate fields.
3. `SetKeywordWithSpecificHandler` – Adds a unique function to the keyword that will be called when that keyword is hit. This handler can be set to multiple keywords, but different handlers may be used for different keywords. For example, you could have "!vote" go to a `CountVote` function, while "!subscribe" goes to the `PlayFanfare` function.

To enable or disable keywords
Keywords can be enabled and disabled using the `ActivateKeywordMatching` command. You could easily turn voting on and off with this command, for example. You can also call `RemoveKeyword` if you no longer need a particular keyword.

**Choosing and using keywords**

A common Twitch pattern is to provide keywords in the format: `!<keyword> <value>`. For example, `!vote yes`.

We recommend that you use this format for your keywords for simplicity, though it is not required. Simply enter your keywords in this format, and enable the system to use them by setting `UseTwitchPattern` to true.

**Troubleshooting**

If keywords don't seem to be working, verify that your incoming keywords match the chosen keyboard format. You can pass the messages through or set a custom handler to help determine if users are acting in an unexpected way, or if the default keyword matching is too limited for your needs.

**Twitch C++ API Reference**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Twitch API and its RESTful functions are represented by two EBus's in Lumberyard:

- `TwitchApiRequestBus`
- `TwitchApiNotifyBus`

**Note**

If you are using the legacy Twitch v5 Gem, the EBus's are the `TwitchRequestBus` and the `TwitchNotifyBus`.

EBus is the interface that's used to make RESTful requests to the Twitch API. For each function in the Twitch API Reference, there is a corresponding call in the `TwitchApiRequestBus` that will queue an HTTP request. The Twitch API Gem also includes additional EBus functions, which are detailed in the following sections.

For general information about using event buses in Lumberyard, see Working with the Event Bus (EBus) system (p. 1851).

Each call made on the `TwitchApiRequestBus` leads to a response you can subscribe to and listen to on the `TwitchApiNotifyBus`. Because calls are made asynchronously, responses are not guaranteed to arrive in the same order as requested. Each response typically includes the following data:

- a unique `TwitchApi::ReceiptId`
- a `TwitchApi::ResultCode`
- data from the JSON response

To determine which response corresponds to a request, you can compare the `ReceiptId` in your requests to the `ReceiptId` in the responses.

To determine if the call was successful, check if `result.m_result` is equal to `ResultCode::Success`. 
The requested data can be found in `result.m_value.m_data`. The TwitchApi Gem handles parsing the JSON responses served by the Twitch API into these data structs. These structs are also reflected in Lumberyard’s BehaviorContext so that they are fully usable from Lua and ScriptCanvas. Note that the structs are not necessarily identical to the JSON response format returned by Twitch. They may have been packed down to reduce layers.

Example handler for `GetExtensionAnalytics`.

```cpp
void OnGetExtensionAnalytics(const ExtensionAnalyticsResult& result) override
{
    if( result.GetId() == sentRequest.GetID() )
    {
        // Display Extension Analytics info. Note this call can return multiple results.
        cout << "Request Twitch Extension Analytics Info" << endl;
        if(result.m_result == TwitchApi::ResultCode::Success)
        {
            // Iterate through all results
            for (const TwitchApi::ExtensionAnalyticsDatum& datum : result.m_value.m_data)
            {
                cout << " Extension Id   : " << datum.extension_id << endl;
                cout << " Type           : " << datum.type << endl;
                cout << " URL            : " << datum.url << endl;
                cout << " Started-At Time: " << datum.started_at << endl;
                cout << " Ended-At Time  : " << datum.ended_at << endl;
            }
        }
        else
        {
            cout << "Failed Result: " << result.m_result << endl;
        }
    }
}
```

`ExtensionAnalyticsResult` is populated from the JSON response received from the Twitch API. The JSON response would have been formatted like this:

```json
{
    "data": [
    {
        "extension_id": "abcdefg",
        "URL": "https://my.twitch.test/123",
        "type": "overview_v1",
        "date_range": {
            "started_at": "2018-04-30T00:00:00Z",
            "ended_at": "2018-05-01T00:00:00Z"
        }
    }
    ]
}
```

Note that the JSON has `started_at` and `ended_at` contained within the `date_range` object. The TwitchApi Gem simplifies the response struct for this call by eliminating `date_range` and placing `started_at` and `ended_at` adjacent to the rest of the data. As a design decision, the TwitchApi Gem tries to keep structs as lean as possible so each response struct may be a leaner version of the JSON response.

**Topics**
- `TwitchApiRequestBus` (p. 2101)
- `TwitchApiNotifyBus` (p. 2105)
TwitchApiRequestBus

The TwitchApiRequestBus includes functions that correspond to their counterparts defined in the Twitch API Reference, plus the following additional functions.

SetApplicationID

Sets the Twitch application ID. This must be called before any other API operation, and should only be called once. You will not receive a notification from this call.

Usage Example

```cpp
/*
** Set a string to hold our Twitch Application ID.
*/
AZStd::string applicationId = "0123456789abcdef0123456789";
TwitchApi::TwitchApiRequestBus::Broadcast(&TwitchApiRequests::SetApplicationId, applicationId);
```

Parameters

applicationID (AZStd::string)

The Twitch application ID that you obtain from Twitch when you create an application.

Return

No return value.

SetAppAccessToken

Sets the Twitch app access OAuth token. When you set the app access token, a new AppAccessTokenNotify event is generated on the TwitchApiNotifyBus EBus and TwitchApi::TwitchApiNotifications::AppAccessTokenNotify() is called.

Usage Example

```cpp
/*
** Set a string to hold our Twitch app access OAuth bearer token.
*/
AZStd::string appToken = "0123456789abcdef0123456789";
TwitchApi::TwitchApiRequestBus::Broadcast(&TwitchApiRequests::SetAppAccessToken, appToken);
```

Parameters

token (AZStd::string)

The Twitch app access OAuth bearer token.

Return

No return value.
**AppAccessTokenNotify Callback**

*token (AZStd::string)*

The Twitch app access OAuth token that was set.

**SetUserAccessToken**

Sets the Twitch user access OAuth token. When you set the user access token, a new `UserAccessTokenNotify` event is generated on the `TwitchApiNotifyBus` EBus and `TwitchApi::TwitchApiNotifications::UserAccessTokenNotify()` is called.

**Usage Example**

```cpp
/*
 ** Set a string to hold our Twitch user access OAuth bearer token.
 */
AZStd::string userToken = "0123456789abcdef0123456789";
TwitchApi::TwitchApiRequestBus::Broadcast(&TwitchApiRequests::SetUserAccessToken, userToken);
```

**Parameters**

*token (AZStd::string)*

The Twitch user access OAuth bearer token.

**Return**

- No return value.

**UserAccessTokenNotify Callback**

*token (AZStd::string)*

The Twitch user access OAuth token that was set.

**SetUserId**

Sets the Twitch user ID. When you set the user ID, a new `UserIdNotify` event is generated on the `TwitchApiNotifyBus` EBus and `TwitchApi::TwitchApiNotifications::UserIdNotify()` is called.

**Usage Example**

```cpp
/*
 ** Set a string to hold our Twitch user ID.
 */
AZStd::string userId = "0123456789abcdef0123456789";
TwitchApi::TwitchApiRequestBus::Broadcast(&TwitchApiRequests::SetUserId, userId);
```
Parameters

**userId (AZStd::string)**

The Twitch User ID. This can be obtained from Twitch by querying a user name.

Return

No return value.

**UserIdNotify Callback**

**userID (AZStd::string)**

The user ID that was set.

**GetApplicationId**

Gets the cached Twitch application Id. There is no notification from this call.

Usage Example

```cpp
/*
 ** Get our Twitch Application ID.
 */
AZStd::string applicationId;
TwitchApi::TwitchApiRequestBus::BroadcastResult(applicationId, &TwitchApiRequests::GetApplicationId);
```

Parameters

No parameters.

Return

**applicationId (AZStd::string)**

The Twitch application ID. This is obtained from Twitch when you create the application.

**GetAppAccessToken**

Gets the cached Twitch app access OAuth token. There is no notification from this call.

Usage Example

```cpp
/*
 ** Get the Twitch app access OAuth bearer token.
 */
AZStd::string oAuthToken;
TwitchApi::TwitchApiRequestBus::BroadcastResult(oAuthToken, &TwitchApiRequests::GetAppAccessToken);
```
Parameters

No parameters.

Return

oAuthToken (AZStd::string)

The Twitch app access OAuth bearer token.

GetUserAccessToken

Gets the cached Twitch user access OAuth token. There is no notification from this call.

Usage Example

```cpp
/*
** Get the Twitch user access OAuth bearer token.
*/
AZStd::string oAuthToken;
TwitchApi::TwitchApiRequestBus::BroadcastResult(oAuthToken, &TwitchApiRequests::GetUserAccessToken);
```

Parameters

No parameters.

Return

oAuthToken (AZStd::string)

The Twitch user access OAuth bearer token.

GetUserId

Gets the cached Twitch user ID. There is no notification from this call.

Usage Example

```cpp
/*
** Get the Twitch user ID.
*/
AZStd::string userId;
TwitchApi::TwitchApiRequestBus::BroadcastResult(userId, &TwitchApiRequests::GetUserId);
```

Parameters

No parameters.
Return

userid (AZStd::string)

The Twitch user ID.

TwitchApiNotifyBus

The TwitchApiNotifyBus includes handler functions that correspond to the request functions found in the TwitchApiRequestBus, including the following callbacks to the request functions defined by the TwitchApi Gem.

AppAccessTokenNotify

Notification event for when the Twitch app access OAuth token is set.

Usage Example

```cpp
// Event gets broadcast.
TwitchApiNotifyBus::QueueBroadcast(&TwitchApiNotifyBus::Events::AppAccessTokenNotify, rAppAccessToken);

// Event handled in override of TwitchApi::TwitchApiNotifications::AppAccessTokenNotify.
void AppAccessTokenNotify(const AZStd::string& appAccessToken)
{
    // Do something with the app access OAuth token.
}
```

Parameters

token (AZStd::string)

The Twitch app access OAuth bearer token.

Return

No return value.

UserAccessTokenNotify

Notification event for when the Twitch user access OAuth token is set.

Usage Example

```cpp
// Event gets broadcast.
TwitchApiNotifyBus::QueueBroadcast(&TwitchApiNotifyBus::Events::UserAccessTokenNotify, rUserAccessToken);

// Event handled in override of TwitchApi::TwitchApiNotifications::UserAccessTokenNotify.
void UserAccessTokenNotify(const AZStd::string& userAccessToken)
{
    // Do something with the user access OAuth token.
}
```

Parameters

token (AZStd::string)

The Twitch user access OAuth bearer token.
Return

No return value.

**UserIdNotify**

Notification event for when the Twitch user ID is set.

**Usage Example**

```cpp
// Event gets broadcast.
TwitchApiNotifyBus::QueueBroadcast(&TwitchApiNotifyBus::Events::UserIdNotify, rUserId);

// Event handled in override of TwitchApi::TwitchApiNotifications::UserIdNotify.
void UserIdNotify(const AZStd::string& userId)
{
    // Do something with the user ID.
}
```

**Parameters**

- `userId` (*AZStd::string*)
  - The Twitch User ID.

**Return**

No return value.

---

**Implementing Connected Features with Cloud Canvas**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

**Topics**

- Features (p. 2107)
- Example Uses (p. 2107)
- Tools (p. 2107)
- Knowledge Prerequisites (p. 2108)
- Cloud Canvas Overview (p. 2108)
- Cloud Gems (p. 2127)
- Cloud Canvas Gameplay Design and Engineering Guide (p. 2221)
- Cloud Canvas Software Engineering Guide (p. 2224)
- Administering Cloud Canvas (p. 2348)
- Using the Cloud Canvas Command Line (p. 2390)

Deeply integrated with AWS, Cloud Canvas is a suite of tools and solutions (cloud gems, resource groups, script canvas nodes) designed to achieve two main goals:
1. Make it easy for you to build cloud-connected features, so you can focus on innovation rather than on undifferentiated backend infrastructure.

2. Make it possible for you to create fantastic new experiences made possible by the availability of the on-demand, global storage and compute provided by AWS.

With Cloud Canvas, you can add a cloud-connected feature to your game in as little as 30 minutes. A single engineer can do this, freeing up the rest of your team to think about innovation and player experience.

Features

Cloud Canvas offers a wide range of helpful features:

- **Cloud gems (p. 2127)** that provide cloud-connected features such as Message of the Day, Leaderboards, and Dynamic Content. These cloud gems can be used in a few clicks as is, or as samples to fuel your custom developments and ideas.
- The **Cloud Gem Framework (p. 2257)**, on top of which cloud gems are built, allows you to add pre-packaged, cloud-connected features in a few clicks. You can use the Cloud Gem Framework to build your own cloud gems (p. 2259).
- Tools to enable a team to build a game with cloud-connected features.
- A **CloudGemAWSScriptBehaviors (p. 2289)** gem that exposes AWS services such as Amazon S3, Amazon Cognito, AWS Lambda, and HTTP utilities to script.
- Tools to manage AWS resources and permissions that determine how game developers and players access them.
- Management of AWS deployments so that development, test, and live resources are maintained separately.
- Methods for players to be authenticated (anonymous and authenticated). Players can be authenticated from a variety of devices and access their game data by logging in with an Amazon, Facebook, or Google account.

Example Uses

Consider the many ways you can use Amazon Web Services for connected games:

- Store and query game data such as player state, high scores, or world dynamic content: Amazon S3 and DynamoDB
- Trigger events in real time and queue data for background processing: Amazon SQS and Amazon SNS
- Execute custom game logic in the cloud without having to set up or manage servers: AWS Lambda
- Employ a daily gift system that tracks player visits and rewards frequent visits: Amazon Cognito, Amazon S3, DynamoDB, AWS Lambda
- Present a message of the day or news ticker that provides updates on in-game events: Amazon Cognito, Amazon S3, AWS Lambda

For tutorials on Cloud Canvas, see Tutorial: Getting Started with Cloud Canvas (p. 2116) and Lumberyard Tutorials.

Tools

You can access Cloud Canvas functionality by using any of the following:

- **Cloud Canvas C++ APIs** – For software development.
Knowledge Prerequisites

You need the following to take advantage of Cloud Canvas:

- **An understanding of AWS CloudFormation Templates** – Cloud Canvas uses the AWS CloudFormation service to create and manage AWS resources. Our goal is for Cloud Canvas to minimize what you need to know about AWS CloudFormation and AWS in general.
- **Familiarity with JSON** – Cloud Canvas leverages JSON for storing configuration data, including AWS CloudFormation Templates. Currently, you'll need to be familiar with this text format to work with the Cloud Canvas resource management system. A JSON tutorial can be found [here](#).

Cloud Canvas Overview

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloud Canvas helps you manage cloud resources and connect your game with the AWS cloud. Understanding its concepts will benefit anyone on your team who interacts with the cloud-connected components of your game, including designers, programmers, and testers.

This section covers the following:

- What Cloud Canvas is and how it relates to your AWS account
- The Amazon Web Services that Cloud Canvas supports
- How Cloud Canvas helps you manage your resources
- How your game can communicate with the cloud through cloud gems

Prerequisites

Before reading this topic, you should have a basic understanding of the Lumberyard engine (p. 1).

AWS, Cloud Canvas, and Lumberyard

Amazon Web Services (AWS) is an extensive and powerful collection of cloud-based services. You can use these services to upload or download files, access databases, execute code in the cloud, and perform many other operations. A cloud service saves you the trouble of maintaining the infrastructure that it relies on.

Cloud-Based Resources

When you want to use an AWS cloud service, you do so through a resource, a cloud-based entity that is available for your use, help, or support. Resources include a database, a location for storing files, the code that a service runs, and more.

When you create a resource, it exists in the cloud, but you can use it and manage its content. You also specify the permissions that individuals or groups have to access or use the resource. For example, you might allow anyone in the public to read from your database but not write to it or modify it.
Resource Groups

In order to create a connected game feature such as a high score table, you create a resource group in Cloud Canvas. The resource group defines the AWS resources that your feature requires. Each connected game feature therefore is implemented as a resource group in Cloud Canvas.

AWS Accounts

Your resources are owned by an AWS account. The AWS account allows you and your team to share access to the same resources. For example, your team's AWS account might own a database resource so that you and your teammate can both work with the same database.

You, or someone on your team, is an administrator. The administrator creates the AWS account for your team and gives individuals on the team access to the account's resources.

Lumberyard, Cloud Canvas, and Cloud Gems

Cloud Canvas is a Lumberyard Gem (extension) that enables your Lumberyard games to communicate with AWS resources. To integrate the communication with Amazon Web Services directly into your game logic, you can use cloud gems (p. 2127).

For example, you can use cloud gems to create leaderboards, messages of the day, in-game surveys, and speech recognition and text-to-speech capabilities for your game.

Amazon Web Services Supported by Cloud Canvas

Several AWS offerings are available through Cloud Canvas that can enhance your game.

File Storage in the Cloud

For storing files in the cloud, Cloud Canvas supports Amazon Simple Storage Service (Amazon S3). Amazon S3 offers a storage resource called a bucket, which you can think of as a large folder. You can build a directory structure in an Amazon S3 bucket just like a directory on a local computer. Amazon S3 buckets have a number of uses in games, including the following:

- Storing files that your game can download. These files can be levels, characters, or other extensions for your game. You can add new files after your game has shipped. Because your game uses Cloud Canvas to download and integrate this content, your customers do not need to download a new client.
- Your game can upload player-generated content. For example, your game might take a screenshot whenever a player beats the last boss. Cloud Canvas uploads the screenshot to your bucket, and your game makes the screenshot available on a website or to other players of the game.

Databases

For storing data like a person's name, phone number, and address in the cloud, Cloud Canvas supports the Amazon DynamoDB database service. Amazon DynamoDB operates on resources called tables. These tables grow and adapt as you build and iterate your game.

Here are some ways in which you can use Amazon DynamoDB table resources in your game:

- Track account details and statistics for a player. Give each player a unique ID so that you can look up a player's hit points, inventory, gold, and friends.
- Add or remove fields to accommodate new resource groups in your game.
- Perform data analyses. For example, you can run complex queries to find out how many players have unlocked a particular achievement.
- Manage game-wide resource groups or events such as a querying a high score table or retrieving a message of the day.
Executing Cloud-Based Logic

For executing code in the cloud, Cloud Canvas supports the AWS Lambda service. AWS Lambda executes resources called functions. You provide the code for a Lambda function, and your game calls the Lambda service through Cloud Canvas to run the function. The Lambda service returns the data from the function to the game.

Your Lambda functions can even call other Amazon Web Services like Amazon DynamoDB and perform operations on their resources. Following are some examples:

- **Submit a high score** – A Lambda function can accept a player's ID and new score, look up the player ID in the database, compare the score with existing scores, and update the highest score if necessary.
- **Sanitize your data** – A Lambda function can check for malicious or unusual input. For example, if a player tries to upload a new high score of 999,999,999 when the best players can't reach 1,000, your Lambda function can intercept the submission and either reject it or flag it for review.
- **Perform server-side authoritative actions** – Cloud Canvas can call your Lambda functions to control in-game economies. For example, when a player tries to purchase an item, your Lambda function can check a database to verify that the player has enough money to pay for the item. The function can then deduct the amount from the player's account, and add the item to the player's inventory.

Identity and Permissions

For managing the identity of the player and controlling access to AWS resources in the cloud, Cloud Canvas supports the Amazon Cognito service.

Amazon Cognito can create unique anonymous identities for your players that are tied to a particular device. It can also authenticate identities from identity providers like Login with Amazon, Facebook, or Google. This provides your game with a consistent player IDs that can seamlessly transition from anonymous use on a single device to authenticated use across multiple devices. Consider these examples:

- Players start playing your game anonymously and store their progress locally on their device. Later, to "upgrade" their experience, you ask them to be authenticated through one of the login providers mentioned. After players provide an authenticated ID, you can store their progress in the cloud, and they can access their progress across multiple devices.
- You can specify which AWS resources players are allowed to access. For example, you can enable the "Get the Latest High Scores" Lambda function to be called not only by your game, but by anyone, including external websites. But you could specify that the "Submit High Scores" function only be called by players of your game so that your high score table remains secure. You can use Cloud Canvas to manage these permissions.

Understanding Cloud Canvas Resource Manager

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Game development is an inherently local activity. You have a local copy of your game code, assets, and other resources. You build, test, and tweak over and over on your local computer.

The cloud is different. It is an alien environment. You put resources "out there" that the game depends on. But those resources don’t live on your computer system. The process of using and modifying the resources in the cloud isn’t the same as for resources that are local.

Cloud Canvas Resource Manager bridges this gap. It lets you have local descriptions of the AWS resources in the cloud that your game needs and provides ways to create and interact with the actual
instances of those resources in AWS. Your resource could be a database table, a file storage bucket, or code that runs in response to an event.

For team projects, the source code and assets that you are using likely come from a source control system. The changes you make are shared with other people who work on the project through that source control system. Different people can be working at the same time with different versions (“branches”) of the code and with different versions of assets without interfering with each other.

When you develop a game that uses cloud resources in AWS, those resources may be shared by different people who work on the game at the same time. Sometimes you need different versions of those resources to exist in the cloud. You also want to ensure that the people developing the game use the version of the resources in the cloud that matches the version of the code and assets they are working with.

After the game is released, the players will use a production copy while your team uses another, private copy to work on bug fixes and new content.

You’ll also want to do the following:

- Be sure that players cannot access the development versions of game resources
- Prevent the development team from making changes that could break the released game
- Protect player information like e-mail addresses from unauthorized access by team members

The Cloud Canvas Resource Manager provides the tools you need to do the following:

- Maintain descriptions of the AWS resources that your game depends on
- Create as many copies of the AWS resources as needed for your releases and development teams
- Help you secure access to those resources

The Role of AWS CloudFormation

The Cloud Canvas Resource Manager integrates the use of AWS CloudFormation into the Lumberyard game development environment. With AWS CloudFormation you can maintain descriptions of the AWS resources you need in text file templates that you can check into your source control system.
descriptions can be branched and merged along with the rest of your game code and assets. When you need actual instances of the resources to be created in AWS, **Cloud Canvas Resource Manager** passes the descriptions to AWS CloudFormation, which uses the template files to create, update, or delete resources in AWS to match the descriptions.

You can use resource manager to organize your descriptions into any number of **resource groups**. Each group can describe all the resources needed by a game feature, such as a high score tracking system. For details, see [Resource Definitions](#p.2236).

With resource manager you can create as many **deployments** of the resources as you need. You could have a deployment for the dev team, another for the QA team, and another for the released game, or any other arrangement that suits your needs. Each deployment contains a complete and independent instance of all of the project's resources. Deployments are implemented using AWS CloudFormation **stack** resources. For details, see [Resource Deployments](#p.2247).

You can choose the deployment that you want to work with in Lumberyard Editor. For example, if you create a "QA" deployment and use it to test your game, Lumberyard Editor automatically maps the references to resources in your game code to the "QA" instance of those resources.

Similarly, you can also specify the deployment to be used for release builds of the game. For details, see [Resource Mappings](#p.2250).

Each deployment comes with an AWS managed policy and an AWS role that you can use to grant specific AWS users and groups access to that deployment. For example, players are granted access to specific resources within a deployment. For details, see [Controlling Access to Resources](#p.2370).

**A Closer Look at AWS CloudFormation Stacks**

A Cloud Canvas Resource manager project consists of one or more AWS CloudFormation stacks. All stack templates are stored as files in the project's source control system.

- A single **project stack** that contains the resources that support Cloud Canvas Resource Manager itself. The project stack template is stored as a file in the project's source control system.

- **Any number of deployment stacks**. A deployment represents a complete and independent set of all the resources needed by the game. Each deployment stack contains a child stack for each resource group. All deployment stacks are defined using a single AWS CloudFormation stack template.
• One deployment access stack per deployment stack. A deployment's access stack defines the resources used to grant access to a deployment, including IAM Roles and Amazon Cognito Identity Pools. They are distinct from deployment stacks because game developers need to be able to update deployment stacks but not change the permissions associated with the deployment.

• Any number of resource group stacks. Each resource group represents a set of resources related to an arbitrary game feature. Resource group stacks exist only as children of deployment stacks. Each resource group has its own stack template, which defines the resource group's stack for each deployment.

So, if there are 3 deployments and 4 resource groups, you have a total of 12 resource group stacks, 3 deployment stacks, 3 deployment access stacks, and 1 project stack (19 stacks total). You'll also have a total of 7 stack templates, one for the project stack, one for all the deployment stacks, one for all the deployment access stacks, and one each for each resource group. The following image illustrates this scenario.

Cloud Canvas Resource Management

In addition to communicating with Amazon Web Services, Cloud Canvas can also help you manage your resources. Amazon Web Services can help create and manage any cloud resources that a game resource group needs. Once you implement the resource group you can use Cloud Canvas deployments to manage the resources for development, test, and live versions of your game.

Defining the Resources

You can create cloud resources by using AWS CloudFormation templates. AWS CloudFormation is an Amazon Web Service with which you can define, create, and manage AWS resources predictably and repeatedly by using templates. The templates are JSON-formatted text files that you use to specify the collection of resources that you want to create together as a single unit (a stack).
In a template, each resource gets its own AWS CloudFormation definition in which you specify the parameters that govern the resource. AWS CloudFormation templates are beyond the scope of this topic, but for now it’s enough to understand that you can define (for example) a template with an Amazon DynamoDB table and two AWS Lambda functions. For an example AWS CloudFormation template that creates an Amazon DynamoDB table, see the AWS CloudFormation User Guide.

Deployments

While you are working on a new resource group, your quality assurance team might have to test it. You want to provide a version of your resource group that the test team can use while you continue to work on your own version. To keep the corresponding resources of the different versions distinct, Cloud Canvas gives you the ability to create separate deployments. Deployments are distinct instances of your product’s features.

In a scenario like the one described, you might create three deployments: one for the development team, one for the test team, and one for live players. Each deployment’s resources are independent of each other and can contain different data because (for example) you don’t want the data entered by the test team to be visible to players.

With Cloud Canvas you can manage each of these deployments independently of one another, and you can switch between deployments at will. After making changes, you can use Cloud Canvas to update your feature or deployment and update the corresponding AWS resources.

Team Workflow Using Deployments

The following workflow example illustrates how Cloud Canvas deployments work:

1. The test team finds a bug. You fix the bug in your Lambda code.
2. You switch to the dev deployment and upload the new version of the Lambda function. The Lambda code in the test and live deployments remain untouched for now, and they continue working as is.
3. After you are satisfied that the bug has been fixed, you update the Lambda code in the test deployment. The test team can now test your fix. The live deployment continues unchanged.
4. After the test team approves the fix, you update the live deployment, propagating the fix to your live players without requiring them to download a new version of the game.

Managing Permissions Using Cloud Canvas

Managing permissions is an important part of building a secure cloud-connected game. Maintaining separate and distinct permissions is important in the phases of development, testing, and production. You can apply permissions to your development and test teams, to the AWS resources that your game uses, and to the players of your game. A key objective is to secure your game’s AWS resources against hackers and other forms of abuse.

You can use permissions to specify exactly who is allowed to do what to the AWS resources that are part of your game. For example, if you have a game feature that uploads screenshots, you can create an Amazon S3 bucket to store the screenshots. You can set permissions for the game to be able to write (send files) to the bucket, but not read from the bucket. This prevents inquisitive users from examining the files that have been uploaded. On the other hand, you can give your team members permissions to read files from the bucket so that they can review and approve them. With Cloud Canvas you can also set the permissions for individual deployments. For example, live and test deployments can have different permission sets.

Like features, you can define permissions through AWS CloudFormation templates. The permissions are applied any time that you update your cloud resources using the Cloud Canvas resource management tools.

For more information, see Controlling Access to Resources (p. 2370).
Cloud Gems Overview

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A Lumberyard gem (p. 1064) is an individual package of specific functionality and assets. The gem includes everything required for you to include that functionality in your project.

Lumberyard cloud gems are Lumberyard gems (p. 1064) that provide AWS functionality for a game. A more complete definition of a cloud gem might be the following:

A cloud gem is an individual package of specific cloud-connected functionality, assets and AWS resource definitions. A cloud gem includes everything required for you to include that cloud-connected functionality in your project.

Cloud Gems Included with Lumberyard

Lumberyard includes many cloud gems that you can enable for your game project. For a list of the cloud gems that are included with Lumberyard, see Cloud Gems (p. 2127). For information on enabling cloud gems in your project, see Enabling Gems (p. 1064).

Cloud Gem Framework

The Cloud Gem Framework is a software development kit (SDK) that is included with Lumberyard. You can use the framework to implement the cloud gems that come with Lumberyard, as well as your own cloud gems. For more information, see Cloud Gem Framework (p. 2257).

Pricing

Cloud Canvas uses AWS CloudFormation templates to deploy AWS resources to your account. Although there is no additional charge for Cloud Canvas or AWS CloudFormation, charges may accrue for using the associated AWS services. You pay for the AWS resources created by Cloud Canvas and AWS CloudFormation as if you created them manually. You only pay for what you use as you use it. There are no minimum fees and no required upfront commitments, and most services include a free tier.

For pricing information on the AWS services that Cloud Canvas supports, visit the following links.

Amazon Cognito Pricing
Amazon DynamoDB Pricing
AWS Lambda Pricing
Amazon S3 Pricing
Amazon SNS Pricing
Amazon SQS Pricing

To see pricing for all AWS services, visit the Cloud Services Pricing page.
Tutorial: Getting Started with Cloud Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloud Canvas connects Lumberyard with Amazon Web Services (AWS) and uses cloud gems to enable connected features in your game. This tutorial shows you how to deploy the resources for a cloud gem enabled project to AWS and guides you through the following tasks:

- Signing up for an AWS account.
- Creating an AWS Identity and Access Management (IAM) user to administer your Cloud Canvas project.
- Enabling cloud gems in your project.
- Adding your IAM administrative credentials to Lumberyard.
- Uploading resources to AWS and creating a deployment.

Optionally, you can also learn how to inspect your resources in AWS, create IAM users and groups for administrative purposes, and remove Cloud Canvas deployments and resources from AWS.

Prerequisites

Before starting this tutorial, complete the following:

- Install and set up (p. 7) Lumberyard.
- Read Implementing Connected Features with Cloud Canvas (p. 2106).

Step 1: Sign up for AWS

When you sign up for AWS, you can access all its cloud capabilities. Cloud Canvas creates resources in your AWS account to make these services accessible through Lumberyard. You are charged only for the services that you use. If you are a new AWS customer, you can get started with Cloud Canvas for free. For more information, see AWS Free Tier.

If you or your team already have an AWS account, skip to Step 2.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Note

- You must provide a payment method in order to create your account. Although the tutorials here fall within the AWS Free Tier, be aware that you can incur costs.
- Make a note of your AWS account number, which you will use in the next step.
Step 2: Create an IAM User to Administer the Cloud Canvas Project

After you sign up for an AWS account, you need an IAM user with appropriate permissions to administer a Cloud Canvas project. IAM allows you to manage access to your AWS account.

AWS requires that you provide credentials to verify that you have the appropriate permissions for the AWS services that you access. You enter these credentials into Lumberyard Editor as part of setting up your project.

The IAM user that you create will belong to a group that has administrator permissions. This allows users in this group to create the Cloud Canvas resources and make them accessible through Lumberyard. Administrative users in this group will have special permissions beyond the scope of a normal Cloud Canvas user.

In a team environment, you—as a member of the administrator's group—can create IAM users for each member of your team. With IAM, you can set permissions specifically for each person's role in a project. For example, you might specify that only designers can edit a database, or prevent team members from accidentally writing to resources with which your players interact.

For more information on IAM and permissions, see the IAM User Guide.

Create an IAM User and an Administrator Group

This section guides you through IAM best practices by creating an IAM user and an administrator group in your account to which the IAM user belongs.

To create an IAM user and group in your account

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, click Users.
3. Click Add user.
4. For User name, enter a user name like CloudCanvasAdmin. The name can contain letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (-), and hyphen (-). The name is not case sensitive and can be a maximum of 64 characters.
5. Select the check box next to Programmatic access.
6. Select the check box next to AWS Management Console access, select Custom password, and then enter a new password in the text box.
   
   Note
   When you create a user for someone else, you can select Require password reset so that the user must create a new password when the user first signs in.

7. Click Next: Permissions.
8. Click Create group.
9. In the Create group dialog box, enter a name like CloudCanvasAdministrators for the group. The name can contain letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (-), and hyphen (-). The name is not case sensitive and can be a maximum of 128 characters.
10. In the Policy name list, select the check box next to AdministratorAccess. This policy provides the necessary permissions to create and administer a Cloud Canvas project.
   
   Warning
   The AdministratorAccess policy allows almost all permissions within the AWS account and should be attached only to the administrator of the account. Otherwise, other team members can perform actions that incur unwanted charges in your AWS account.

11. Click Create group.
12. In the list of groups, select the check box for your new group if it is not already selected. If necessary, click Refresh to see the group in the list.
13. Click **Next: Review** to review your choices.
14. When you are ready to proceed, click **Create user**.

Your IAM user is created along with two important credentials: an access key and a secret access key. Later, you enter these credentials into Lumberyard Editor to access the AWS resources in your project.

15. Click **Show** to view your secret access key and password, or click **Download .csv** to download the credentials in a .csv file. Your credentials will look something like this:

- Access key ID: AKIAIOSFODNN7EXAMPLE
- Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

You can also click **Send email** to receive login instructions by email. After this point, you cannot view the secret access key from the AWS Management Console.

**Important**
Keep the keys confidential in order to protect your AWS account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

16. You have now created an IAM user called **CloudCanvasAdmin** and a **CloudCanvasAdministrators** administrator group to which the user belongs. To confirm this, click **Groups** in the navigation pane. Under **Group Name**, click **CloudCanvasAdministrators**. The **CloudCanvasAdmin** user appears in the list of users for the group.

**Note**
In this tutorial, you add only one IAM user to the administrator group, but you can add more if required.

If you lose your secret access key, you must create a new set of keys.

**To create a new set of keys**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, click **Users**.
3. In the **User name** list, click the user name for which you want to generate new access keys.
4. On the **Summary** page, click the **Security credentials** tab.
5. In the **Access keys** section, click **Create access key**.

**Step 3: Enable One or More Cloud Gems in Your Project**

Cloud Canvas functionality is enabled in Lumberyard through gems (p. 1064) and cloud gems (p. 2127). Gems are extensions that share code and assets among Lumberyard projects. Cloud gems are gems that use the power of AWS to provide connected features. To enable gems and cloud gems in your project, you use the **Project Configurator** (p. 43).

This tutorial uses the Lumberyard CloudGemSamples project, which includes many of the cloud gems that come with Lumberyard.

**To enable the CloudGemSamples project**

1. Open the Lumberyard Project Configurator, located at `lumberyard-version\dev\Bin64\BuildPlatform\ProjectConfigurator.exe`. For example, when using Visual Studio 2017 as your build platform, the Project Configurator is located at `lumberyard-version\dev\Bin64\vc141\ProjectConfigurator.exe`. 
2. Click **CloudGemSamples**.
3. In the upper-right corner of the Project Configurator, click **Set as default**.

**Step 4: Add Administrator Credentials to Lumberyard**

In order to begin managing a Cloud Canvas project, you add the IAM user credentials that you generated earlier to a profile that Cloud Canvas can easily reference. To do this, you can use either Lumberyard Editor or a command line prompt.

**To enter your credentials in Lumberyard Editor**

1. Use one of the following methods to launch Lumberyard Editor:
   - From the desktop, double-click the Lumberyard Editor icon.
   - From Lumberyard Setup Assistant, on the **Summary** page, click **Launch Editor**.
   - For Visual Studio 2017, navigate to the `lumberyard-version\dev\Bin64vc141` directory and double-click `Editor.exe`.
2. In Lumberyard Editor, choose **AWS, Credentials manager**.
3. In the **Credentials Manager** dialog, click **Add profile**.
4. In the **Add profile** dialog box, enter the following information:
   - For **Profile name**, enter a name of your choice (for example, **CloudCanvasAdmin**).
   - For **AWS access key** and **AWS secret key**, enter the access key and secret key.
Important
Do not share these credentials with anyone, and do not check them into source control. These credentials grant control over your AWS account, and a malicious user could incur charges.

5. Click Save.
6. In Credentials Manager, click OK.

You have now created a profile to administer a Cloud Canvas project. The profile name is associated with your credentials, and saved locally on your machine in your AWS credentials file. This file is normally located in your C:\Users\user_name\.aws\ directory. As a convenience, other tools such as the AWS Command Line Interface or the AWS Toolkit for Visual Studio can access these credentials.

Step 5: Upload Resources to AWS and Create a Deployment

You are now ready for the resources defined by your project to be created in AWS.

This is a two-part process:
1. Creating a project stack in AWS that contains the resources that your cloud gems require.
2. Creating a deployment. A deployment is a separate, independent copy of your project's AWS resources for a dedicated purpose (for example, development, testing, or production).

To upload resources to AWS

1. In Lumberyard Editor, choose **AWS, Cloud Canvas, Resource Manager**.
2. In **Resource Manager**, click **Upload all resources**.

3. In the **Initialize Resource Manager** dialog box, click **Yes**.

4. In the **Initialize Cloud Canvas Resource Manager** dialog box, review the fields provided.
Overview

• **Project stack name** – This name defaults to the name of your project. You can provide a different name if you want.

• **AWS profile** – This is the profile that you prepared in the previous procedure. You can click **Edit** to edit this profile or **Add profile** to add another profile.

• **AWS region** – This defaults to **us-east-1**, which supports all Lumberyard cloud gems. Not all cloud gems are supported in all AWS regions.

5. Click **Create**. The **Progress log** indicates **Operation in progress**. Creating the project stack in AWS takes about 5 to 10 minutes.

6. In the **Create deployment** dialog box, enter a name for your deployment (for example, **dev**, **test**, or **prod**).
7. Click **Create**. The deployment operation takes about 10 minutes.

**Step 6: Learn More**

See the following resources.

<table>
<thead>
<tr>
<th>Task</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn more about each of the cloud gems (including cloud gems not in the CloudGemSamples project)</td>
<td>Cloud Gems (p. 2127)</td>
</tr>
<tr>
<td>Add additional gems and cloud gems to a project</td>
<td>Enabling Gems (p. 1064)</td>
</tr>
<tr>
<td>Use the AWS Management Console to see your resources in AWS</td>
<td>Inspect Your Resources in AWS (p. 2124)</td>
</tr>
</tbody>
</table>
Inspect Your Resources in AWS

This topic shows you how to sign in to the AWS Management Console and use the console to inspect the AWS CloudFormation stacks that you created.

To sign in to the AWS Management Console as your IAM user

1. Retrieve the AWS account ID that you received when you created your AWS account. To sign in as your CloudCanvasAdmin IAM user, use this AWS account ID.
2. In a web browser, enter the following URL with your account ID:

   https://My_AWS_Account_ID.signin.aws.amazon.com/console/

   For example, if your AWS account ID is 1234-5678-9012, you sign in at https://123456789012.signin.aws.amazon.com/console/.

   For convenience, you can create a bookmark of your URL for future use.

   **Tip**
   To create a bookmark for your account sign-in page in your web browser, you should manually type the sign-in URL for your account in the bookmark entry. Do not use your web browser bookmark feature because redirects can obscure the sign-in URL.
3. Enter the CloudCanvasAdmin IAM user name that you created.
4. Enter the password for the user and choose **Sign In**.

   You are now successfully signed into the AWS Management Console.

To inspect your resources in AWS

1. Ensure that the AWS region, which appears on the upper-right of the console screen, is set to the region that you specified when you had Cloud Canvas deploy its resources in Step 5 (p. 2120) of the tutorial. This tutorial uses the US East (N. Virginia) Region.
2. Do one of the following:
   - In the AWS services search box, enter CloudFormation, and then click CloudFormation.
   - Expand All services and under Management Tools, click CloudFormation.
3. On the Stacks page, you can see the individual stacks that have been created for your cloud gems and for your deployment.
Overview

Note
If a stack update operation is still in progress, the stack shows the status UPDATE-IN-PROGRESS. Otherwise, the status shows CREATE-COMPLETE. You can click the Refresh icon to update the status.

Create IAM Users and Groups to Administer Cloud Canvas Teams

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To manage your team members' access to AWS resources, you create IAM users, a group for the users, and then attach the appropriate Cloud Canvas customer managed policies to the group. The policies that Cloud Canvas creates for your IAM users are more restrictive than those for an administrator. You can use these policies so that your team members do not inadvertently incur charges without administrator approval.

To create IAM users and a user group with managed policies

1. Sign in to the AWS Management Console with your CloudCanvasAdmin credentials. See Inspect Your Resources in AWS (p. 2124).
2. Do one of the following:
   - In the AWS services search box, enter IAM, and then click IAM.
   - Expand All services and under Security, Identity & Compliance, click IAM.
3. In the navigation pane, click Users.
4. Click Add user.
5. On the Add user page, in the User name box, enter an IAM user name for a team member.
   The user name can contain of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (-), and hyphen (-). The name is not case sensitive and can be a maximum of 64 characters.
6. For each additional user, click Add another user, and then enter a user name.
7. Under Select AWS access type, select Programmatic access and/or AWS Management Console access, depending on the access that you want each user to have.
If you choose AWS Management Console access, the Console password and Require password reset options appear.

- **Console password** – Enter a password or choose to have a password generated automatically (default).
- **Require password reset** – Requires your users to create a password at their next sign in. This is selected by default.

8. Click **Next: Permissions**.

9. Click **Create group** to create an IAM group for your new users.

10. On the **Create group** page, for **Group name**, enter a name for the group (for example, *CloudCanvasDevelopers*).

    The group name can contain letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (-), and hyphen (-). The name is not case sensitive and can be a maximum of 128 characters.

11. To find the IAM customer managed policy that Cloud Canvas created for you, click **Filter policies** and then select **Customer managed**.

12. Select the appropriate policies that you want to apply to the group. To decide which policies to include, use the information in the following table.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeploymentAccess</td>
<td>Grants the user read-only access to information from a specific deployment stack. Mainly used to describe information for resources used in the stack.</td>
</tr>
<tr>
<td>DeploymentOwnerAccess</td>
<td>Grants full access to the deployment stack.</td>
</tr>
<tr>
<td>ProjectAccess</td>
<td>Grants a user read-only access to the configuration of the project stack.</td>
</tr>
<tr>
<td>ProjectOwnerAccess</td>
<td>Grants full access to the project stack and deployments in that stack.</td>
</tr>
<tr>
<td>ServiceLambdaExecution</td>
<td>Do not use.</td>
</tr>
</tbody>
</table>

13. Click **Create group**.

14. Click **Next: Review**. The **Review** page lists the user names that you created and summarizes the permissions that are to be granted.

15. Click **Create users**.

    After you successfully created your users, you can view the access key ID, secret access key, and automatically generated password for each new user. You can also email users instructions to sign in to the AWS Management Console.

16. Click **Download .csv** to download a .csv file that has the keys for all the users that you created. Make sure you preserve the credentials in a safe place.

    **Important**
    - After this point, you cannot view the secret access key from the AWS Management Console. You must deliver each user his or her keys securely.
As an administrator, it is your responsibility to keep your team and your AWS account secure. Amazon provides some best practices and options for how to manage your team’s access keys on the Managing Access Keys for IAM Users page in the IAM User Guide. You are encouraged to read this thoroughly.

17. Click Close.

Next, have your team members use Lumberyard Editor to create their AWS profiles with the credentials that you provided.

**To have users create AWS profiles**

1. In Lumberyard Editor, have each user choose AWS, Credentials manager.
2. Have the user enter a new profile name and his or her access and secret access keys.

   **Important**
   Stress the importance to your users of keeping their keys secure and not sharing them.

For information regarding limits on the number of groups and users in an AWS account, see Limitations on IAM Entities and Objects in the IAM User Guide.

**Cloud Gems**

Some cloud Gems have been deprecated and are no longer supported.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A cloud gem is an individual package of specific cloud-connected functionality, assets, and AWS resource definitions. A cloud gem includes everything required for you to add cloud-connected functionality to your project.

Lumberyard includes the following cloud gems. You can enable them in a project from the Lumberyard Project Configurator (p. 43) tool.

- **Compute Farm** (p. 2154) (deprecated) – Divide and conquer large-scale tasks for processing on a fleet of Amazon EC2 instances.
- **Defect Reporter** (p. 2155) (deprecated) – Streamlines and customizes the collection and management of defect report data from users.
- **Dynamic Content** (p. 2155) – Allows Lumberyard .pak files that contain new and updated game assets to be uploaded to the cloud for later automatic downloading to the game client.
- **Game Metrics** (p. 2186) (deprecated) – Collects event data on player behaviors for analysis or triggering event actions.
- **In-Game Survey** (p. 2187) (deprecated) – Create surveys for your game, test them, and publish them.
- **Leaderboard** (p. 2188) (deprecated) – Stores a player's high scores and provides leaderboards that show player rankings.
- **Message Of The Day** (p. 2188) (deprecated) – Schedules the delivery of messages (for example, new product announcements or holiday greetings) to game clients.
- **Player Account** (p. 2188) – Provides a standalone player authentication and management solution that uses Amazon Cognito.
Cloud Gems

- **Speech Recognition** (p. 2203) (deprecated) – Add speech recognition and natural language processing to your Lumberyard game. The Speech Recognition Cloud Gem uses the Amazon Lex service, which recognizes and understands spoken input.
- **Text-to-Speech** (p. 2203) (deprecated) – Enhance your gameplay and workflows with synthesized speech. The Cloud Canvas Text-to-Speech (TTS) Cloud Gem uses Amazon Polly, which is a text-to-speech service that turns text into lifelike speech.
- **Web Communicator** (p. 2204) – Send AWS cloud service events to your game’s connected editors or clients. Your editors or game clients can use this information to update themselves without having to poll AWS for updates.

**Topics**
- Using the AWS Script Behaviors Cloud Gem (p. 2128)
- Compute Farm Cloud Gem (p. 2154)
- Defect Reporter Cloud Gem (p. 2155)
- Using Dynamic Content Manager (p. 2155)
- Game Metrics Cloud Gem (p. 2186)
- In-Game Survey Cloud Gem (p. 2187)
- Leaderboard Cloud Gem (p. 2188)
- Message Of The Day Cloud Gem (p. 2188)
- Player Account Cloud Gem (p. 2188)
- Speech Recognition Cloud Gem (p. 2203)
- Text to Speech Cloud Gem (Using Amazon Polly) (p. 2203)
- Web Communicator Cloud Gem (p. 2204)

**Using the AWS Script Behaviors Cloud Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the AWS Script Behaviors cloud gem in Amazon Lumberyard to implement some common AWS behaviors for your game. To implement these behaviors, you can use Script Canvas or Lua script. These behaviors include decoding URLs, performing HTTP GET operations, transferring files to and from Amazon S3, and invoking AWS Lambda functions. For programmatic information regarding the AWS Script Behaviors cloud gem, see AWS Behavior Context Reflections (p. 2289).

**Prerequisites**

To use the features of this cloud gem, perform the following steps.

1. Enable **Cloud Gem AWS Script Behaviors** in Project Configurator. The examples in this section use the **CloudGemSamples** project, which you can also enable in Project Configurator. The AWS Script Behaviors cloud gem is enabled in the **CloudGemSamples** project by default.
2. Open the Cloud Canvas Resource Manager and update your deployment or upload all the resources in the **CloudGemAWSScriptBehaviors** resource group.

**Topics**
- Decoding a URL with Script Canvas (p. 2129)
- Performing HTTP GET with Script Canvas (p. 2132)
- Uploading a File to Amazon S3 with Script Canvas (p. 2136)
Decoding a URL with Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the AWSBehaviorURL Decode node in Script Canvas to decode an encoded URL, as in the following example:

```
```

When the URL is decoded, the escape characters are converted to normal characters, as in the following example:

```
https://docs.aws.amazon.com/lumberyard/latest/userguide/cloud-canvas-intro.html
```

Topics

- Step 1: Add Nodes to Decode a URL (p. 2129)
- Step 2: Add Nodes to Display the Result (p. 2130)
- Step 3: Test the URL Decoder (p. 2131)

Step 1: Add Nodes to Decode a URL

The following procedure shows you how to create a Script Canvas graph that decodes a URL.

To create a graph that decodes a URL node

1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the Node Palette, expand Utilities and drag the On Graph Start node to the canvas.
3. On the right, in Variable Manager, click Create Variable.
4. In the Variable Type search box, enter AWSBehaviorURL, or scroll down to AWSBehaviorURL.
5. Click AWSBehaviorURL. In Node Inspector, AWSBehaviorURL Variable appears. The default name of the variable is Variable 1.
6. In the Node Inspector, expand AWSBehaviorURL to show the URL box. For URL, which is the input field for the variable, enter an encoded URL (for example, http%3A%2F%2Fdocs.aws.amazon.com%2Flumberyard%2Flatest%2Fuserguide%2Fcloud-canvas-intro.html).
7. From the Variable Manager, press Shift and drag Variable 1 AWSBehaviorURL to the canvas to create the Get Variable 1 node. (You can also drag AWSBehaviorURL to the canvas and then select Get Variable from the drop-down menu.)
8. Connect the Out pin of the On Graph Start node to the In pin of the Get Variable 1 node.
9. In the Node Palette, expand AWS, AWSBehaviorURL.
10. Drag the Decode node to the right of the Get Variable 1 node on the canvas.
11. Connect the Out pin of the Get Variable 1 node to the In pin of the Decode node.
12. Connect the AWSBehavioralURL pin of the Get Variable 1 node to the AWSBehavioralURL:0 pin of the Decode node.
Step 2: Add Nodes to Display the Result

At this point, if you saved the graph and started the level, the Get Variable node would pass its value to the Decode node, which would decode it. However, you wouldn't see the result. To see the result, you must add AWSBehaviorURLNotificationBus nodes and a Print node to the graph. These nodes monitor for AWSBehaviorURL events and show you the decoding results in the Lumberyard console window.

**To display the decoded URL in the console window**

1. In the Node Palette, expand AWS, AWSBehaviorURLNotificationBus.
2. Drag OnError to the canvas. Place the node under the three nodes that are already connected.
3. Drag OnSuccess to the canvas and place it under the OnError node.
4. In the Node Palette, expand Utilities, Debug and then drag Print to the right of the two AWSBehaviorURLNotificationBus nodes on the canvas. The Print node displays messages in the Lumberyard Editor console.
5. Connect the Out pins of the AWSBehaviorURLNotificationBus OnError and OnSuccess nodes to the In pin of the Print node.
6. Connect the String pins of the AWSBehaviorURLNotificationBus OnError and OnSuccess nodes to the Value pin on the Print node. Your canvas should look similar to the following image:
7. Save the canvas with a name like `MyUrlDecoder.scriptcanvas`. The default file location is the `lumberyard_version\dev\project_name\scriptcanvas` directory.

8. Exit the Script Canvas editor.

Step 3: Test the URL Decoder

Now you are ready to attach the script canvas to an entity and test it.

To test the URL decoder

1. In Lumberyard Editor, right-click the viewport and choose Create entity.
2. In Entity Inspector, click Add Component.
3. Under Scripting, click Script Canvas to add a Script Canvas (p. 818) component to the entity.
4. Under Script Canvas, click the browse button (…).
5. In the Pick Script Canvas dialog box, choose the canvas that you created – for example, `myurldecoder (Script Canvas)`, and then click OK.
6. If the Lumberyard console window is not already open, press ` or choose Tools, Console to open the console window. If the console window is already open and you want to clear it, press Ctrl+Shift+C.

7. Press Ctrl+G to start the level. The decoded URL appears in the console, as in the following example. The relevant output is bold text.

```
general.enter_game_mode
Returned:
(Found resource management based identity pool %s.) - us-east-1:guid
(Found resource management based identity pool %s for authenticated access.) - us-east-1:guid
(CloudCanvas) - Anonymous Credentials pulled successfully for identity pool us-east-1:guid.
Disable Accelerators
(Script Canvas) - http://docs.aws.amazon.com/lumberyard/latest/userguide/cloud-canvas-intro.html
SetGlobalState 11->2 'LEVEL_LOAD_TEXTURES'->'RUNNING' 69.1 seconds
general.exit_game_mode
```

Performing HTTP GET with Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the AWSBehaviorHTTP Get node in Script Canvas to perform an HTTP GET operation on a URL that you specify.

**Topics**

- Step 1: Add Nodes to Perform an HTTP GET Operation (p. 2132)
- Step 2: Add Nodes to Display the Response (p. 2133)
- Step 3: Add Nodes to Display the Success or Failure of the Operation (p. 2133)
- Step 4: Perform a Test GET (p. 2135)

**Step 1: Add Nodes to Perform an HTTP GET Operation**

The following procedure shows you how to create a Script Canvas graph that performs an HTTP GET operation.

**To create a graph that performs an HTTP GET operation**

1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the Node Palette, expand Utilities and drag the On Graph Start node to the canvas.
3. On the right, in Variable Manager, click Create Variable.
4. In the Variable Type search box, enter AWSBehaviorHTTP, or scroll down to AWSBehaviorHTTP.
5. Click AWSBehaviorHTTP. In Node Inspector, AWSBehaviorHTTP Variable appears. The default name of the variable is Variable 1.
6. In Node Inspector, expand AWSBehaviorHTTP to show the URL box.
7. For URL, which is the input field for the variable, enter a URL for the GET operation (for example, http://example.com/).
8. From the Variable Manager, press Shift and drag Variable 1 AWSBehaviorHTTP to the canvas to create the Get Variable 1 node. (You can also drag AWSBehaviorHTTP to the canvas and then select Get Variable from the drop-down menu.)
9. Connect the **Out** pin of the **On Graph Start** node to the **In** pin of the **Get Variable 1** node.

10. In the **Node Palette**, expand **AWS, AWSBehaviorHTTP**.

11. Drag the **Get** node to the right of the **Get Variable 1** node on the canvas.

12. Connect the **Out** pin of the **Get Variable 1** node to the **In** pin of the **Get** node.

13. Connect the **AWSBehaviorHTTP** pin of the **Get Variable 1** node to the **AWSBehaviorHTTP:0** pin of the **Get** node.

**Step 2: Add Nodes to Display the Response**

To show the response to the HTTP request, you add an **AWSBehaviorHTTPNotificationsBus GetResponse** node, a **ToJSON** node, and **Print** nodes to the graph. The nodes show you the result in the Lumberyard console window.

**To show the response to the HTTP request in the console window**

1. In the **Node Palette**, expand **AWS, AWSBehaviorHTTPNotificationsBus**.
2. Drag **GetResponse** to the canvas. Place the node under the three nodes that are already connected.
3. In the **Node Palette**, expand **Utilities, String Map** and then drag **ToJSON** to the right of the **GetResponse** node.
4. Connect the **Out** pin of the **GetResponse** node to the **In** pin of the **ToJSON** node.
5. Connect the **StringMap** pin of the **GetResponse** node to the **StringMap: 0** pin of the **ToJSON** node.
6. In the **Node Palette**, expand **Utilities, Debug** and then drag four **Print** nodes to the right of the **ToJSON** node on the canvas and align them vertically. **Print** nodes display messages in the Lumberyard Editor console. After you connect them, each **Print** node will display a different part of the GET response in the console.
7. Connect the **Out** pin of the **ToJSON** node to the **In** pin on the first **Print** node.
8. Connect the **Result: String** pin of the **ToJSON** node to the **Value** pin on the first **Print** node.
9. Connect the **Out** pin of the **GetResponse** node to the **In** pin of the three remaining **Print** nodes.
10. Connect the **Number** pin of the **GetResponse** node to the **Value** pin of the second **Print** node.
11. Connect the first **String** pin of the **GetResponse** node to the **Value** pin of the third **Print** node.
12. Connect the second **String** pin of the **GetResponse** node to the **Value** pin of the fourth **Print** node.

**Step 3: Add Nodes to Display the Success or Failure of the Operation**

Next, you add nodes to display the success or failure of the GET operation.

**To display the success or failure of the GET operation in the console window**

1. In the **Node Palette**, expand **AWS, AWSBehaviorHTTPNotificationsBus**.
2. Drag **OnError** to the canvas. Place the node under the other nodes that are already connected.
3. Drag **OnSuccess** to the canvas and place it under the **OnError** node.
4. From **Node Palette, Utilities, Debug**, drag a **Print** node to the right of the **OnError** and **OnSuccess** nodes.
5. Connect the **Out** pins of the **OnError** and **OnSuccess** nodes to the **In** pin of the new **Print** node.
6. Connect the **String** pins of the **OnError** and **OnSuccess** nodes to the **Value** pin of the new **Print** node. Your canvas should look similar to the following image:
7. Save the canvas with a name like `MyHttpGetTest.scriptcanvas`. The default file location is the `lumberyard_version\dev\project_name\scriptcanvas` directory.

8. Exit the Script Canvas editor.
Step 4: Perform a Test GET

Now you are ready to attach the script canvas to an entity and then test it.

To test HTTP GET

1. In Lumberyard Editor, right-click the viewport and choose Create entity.
2. In Entity Inspector, click Add Component.
3. Under Scripting, click Script Canvas to add a Script Canvas (p. 818) component to the entity.
4. Under Script Canvas, click the browse button (...).
5. In the Pick Script Canvas dialog box, choose the canvas that you created – for example, myhttpgettest (Script Canvas), and then click OK.
6. If the Lumberyard console window is not already open, press ` or choose Tools, Console to open the console window. If the console window is already open and you want to clear it, press Ctrl+Shift+C.
7. Press Ctrl+G to start the level. The console reports the success of the operation and displays the response, as in the following example. The JSON output in the example has been formatted for readability.

```plaintext
general.enter_game_mode
Returned:
(Found resource management based identity pool %s.) - us-east-1:
(Found resource management based identity pool %s for authenticated access.) - us-east-1:
(cloudCanvas) - Anonymous Credentials pulled successfully for identity pool us-east-1:
guid.
Disable Accelerators
(script Canvas) - Success!
(script Canvas) - 200.000000
(script Canvas) - text/html
(script Canvas) - <!doctype html>
<head>
<title>Example Domain</title>
<meta charset="utf-8" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style type="text/css">
body { 
    background-color: #f0f0f2;
    margin: 0;
    padding: 0;
    font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
}

div { 
    width: 600px;
}(script Canvas) -
{ 
    "date": "Mon, 23 Apr 2018 20:42:03 GMT",
    "etag": "\"1541025663+gzip+ident\"",
    "expires": "Mon, 30 Apr 2018 20:42:03 GMT",
    "last-modified": "Fri, 09 Aug 2013 23:54:35 GMT",
    "vary": "Accept-Encoding",
    "x-cache": "HIT",
    "cache-control": "max-age=604800",
    "content-type": "text/html",
    "content-length": "1270",
    "via": "network_information",
    "connection": "keep-alive",
```
Uploading a File to Amazon S3 with Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the AWSBehaviorS3Upload node in Script Canvas to upload a file from your computer to the Amazon S3 bucket that you specify. If the object that you specify already exists, it is overwritten.

Topics
- Step 1: Add Nodes that Upload a File to Amazon S3 (p. 2136)
- Step 2: Add Nodes to Display the Result (p. 2137)
- Step 3: Test Upload a File to Amazon S3 (p. 2138)
- Step 4: Verify the Upload (p. 2139)

Step 1: Add Nodes that Upload a File to Amazon S3

The following procedure shows you how to create a Script Canvas graph that uploads a file to Amazon S3. The example uses the file lumberyard_version\dev\Cache\CloudGemSamples\pc\cloudgemsamples\levels\awsbehaviorexamples\testdata\s3example.txt that is included with Lumberyard.

To upload a file to Amazon S3

1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the Node Palette, expand Utilities and drag On Graph Start to the canvas.
3. On the right, in Variable Manager, click Create Variable.
4. In the Variable Type search box, enter AWSBehaviorS3Upload, or scroll down to AWSBehaviorS3Upload.
5. Click AWSBehaviorS3Upload. In Node Inspector, AWSBehaviorS3Upload Variable appears. The default name of the variable is Variable 1.
6. In Node Inspector, expand AWSBehaviorS3Upload to show the input fields. Type the values in the following table into the corresponding boxes. KeyName is the name that the file will have on Amazon S3.

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BucketName</td>
<td>CloudGemAWSScriptBehaviors.s3nodeexamples</td>
<td>Name of the Amazon S3 bucket for the uploaded file.</td>
</tr>
<tr>
<td>KeyName</td>
<td>uploadtest.txt</td>
<td>Name of the file as it will appear in the Amazon S3 bucket.</td>
</tr>
<tr>
<td>LocalFileName</td>
<td>levels\awsbehaviorexamples\testdata\s3example.txt</td>
<td>Path (relative to the project folder) and name of the file to be uploaded. File path aliases like @user@ are supported. The path resolves to @assets@ by default.</td>
</tr>
<tr>
<td>Input Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>ContentType</strong>: text/html</td>
<td>Content type of the object to be uploaded. Because you are uploading a text file, the content type is text/html.</td>
</tr>
</tbody>
</table>

7. From the Variable Manager, press Shift and drag Variable 1 AWSBehaviorS3Upload to the canvas to create the Get Variable 1 node. (You can also drag AWSBehaviorS3Upload to the canvas and then select Get Variable from the drop-down menu.)

8. Connect the Out pin of the On Graph Start node to the In pin of the Get Variable 1 node.

9. In the Node Palette, expand AWS, AWSBehaviorS3Upload.

10. Drag the Upload node to the right of the Get Variable 1 node on the canvas.

11. Connect the Out pin of the Get Variable 1 node to the In pin of the Upload node.

12. Connect the AWSBehaviorS3Upload pin of the Get Variable 1 node to the AWSBehaviorS3Upload:0 pin of the Upload node.

**Step 2: Add Nodes to Display the Result**

Next, to see the success or failure of the operation, add AWSBehaviorS3UploadNotificationBus nodes and a Print node to the graph. The nodes monitor for AWSBehaviorS3UploadNotificationBus events and show you the result in the Lumberyard console window.

**To display the success or failure of the upload operation in the console window**

1. In the Node Palette, expand AWS, AWSBehaviorS3UploadNotificationBus.

2. Drag OnError to the canvas. Place the node under the three nodes that are already connected.

3. Drag OnSuccess to the canvas and place it under the OnError node.

4. In the Node Palette, expand Utilities, Debug and drag Print to the right of the two AWSBehaviorS3UploadNotificationBus nodes on the canvas. The Print node displays messages in the Lumberyard Editor console.

5. Connect the Out pins of the AWSBehaviorS3UploadNotificationBus OnError and OnSuccess nodes to the In pin of the Print node.

6. Connect the String pins of the AWSBehaviorS3UploadNotificationBus OnError and OnSuccess nodes to the Value pin on the Print node. Your canvas should look similar to the following image:
7. Save the canvas with a name like `MyS3UploadTest.scriptcanvas`. The default file location is the `lumberyard_version\dev\project_name\scriptcanvas` directory.

8. Exit the Script Canvas editor.

**Step 3: Test Upload a File to Amazon S3**

Now you are ready to attach the script canvas to an entity and test the script.

**To test uploading to Amazon S3**

1. In Lumberyard Editor, right-click the viewport and choose Create entity.
2. In Entity Inspector, click Add Component.
3. Under Scripting, click Script Canvas to add a Script Canvas (p. 818) component to the entity.
4. Under Script Canvas, click the browse button (...).
5. In the Pick Script Canvas dialog box, choose the canvas that you created – for example, `mys3uploadtest (Script Canvas)`, and then click OK.
6. If the Lumberyard console window is not already open, press ` or choose **Tools, Console** to open the console window. If the console window is already open and you want to clear it, press **Ctrl+Shift+C**.

7. Press **Ctrl+G** to start the level. The console reports the success of the operation, as in the following example.

   ```
   general.enter_game_mode
   Returned:
   (Found resource management based identity pool %s.) - us-east-1:guid
   (Found resource management based identity pool %s for authenticated access.) - us-east-1:guid
   (CloudCanvas) - Anonymous Credentials pulled successfully for identity pool us-east-1:guid.
   Disable Accelerators
   (Script Canvas) - File Uploaded
   general.exit_game_mode
   ```

**Step 4: Verify the Upload**

To verify that the file has been uploaded to Amazon S3, you can use the AWS Management Console.

**To verify that the test file has been uploaded to Amazon S3**

1. In a text editor, open the `lumberyard_version\dev\Cache\CloudGemSamples\pc\user\AWS\user-settings.json` file.

2. In the `user-settings.json` file, locate the Amazon S3 name for the `CloudGemAWSScriptBehaviors.s3nodeexamples` bucket. The Amazon S3 bucket name is in the string for the `PhysicalResourceId` attribute, as in the following example:

   ```json
   "CloudGemAWSScriptBehaviors.s3nodeexamples": {
      "PhysicalResourceId": "cgsamples14-221-cgsamples14-221dep-s3nodeexamples-16ud5gt53zjx7",
      "ResourceType": "AWS::S3::Bucket"
   }
   ```

3. In Lumberyard Editor, choose **AWS, Open AWS Console, S3**.

4. In the Amazon S3 management console, open the bucket that you identified in step 2.

5. The `s3uploadtest.txt` file appears in the bucket.

**Downloading a File from Amazon S3 with Script Canvas**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **AWSBehaviorS3Download** node in Script Canvas to download a file from an Amazon S3 bucket.

**Topics**

- Step 1: Prepare a Test File for Download (p. 2140)
- Step 2: Add Nodes to Download a File from Amazon S3 (p. 2140)
- Step 3: Add Nodes to Display the Result (p. 2141)
- Step 4: Test Download the File from Amazon S3 (p. 2142)
Step 1: Prepare a Test File for Download

The following procedure shows you how to upload a file to Amazon S3 that you can later download with a Script Canvas script.

To prepare a test file for download from Amazon S3

1. In a text editor, create a plain text file and save it as downloadtest.txt.
2. In a text editor, open the `lumberyard_version\dev\Cache\CloudGemSamples\pc\user\AWS\user-settings.json` file.
3. In the `user-settings.json` file, locate the Amazon S3 name for the `CloudGemAWSScriptBehaviors.s3nodeexamples` bucket. The Amazon S3 bucket name is in the string for the `PhysicalResourceId` attribute, as in the following example:

   ```json
   "CloudGemAWSScriptBehaviors.s3nodeexamples": {
     "PhysicalResourceId": "cgsamples14-221-cgsamples14-221dep-s3nodeexamples-16ud5gt53zjx7",
     "ResourceType": "AWS::S3::Bucket"
   }
   ```

4. In Lumberyard Editor, choose AWS, Open AWS Console, S3.
5. In the Amazon S3 management console, open the bucket that you identified in step 3.
6. Click Upload to upload the downloadtest.txt to the bucket that is mapped to `CloudGemAWSScriptBehaviors.s3nodeexamples`.

Step 2: Add Nodes to Download a File from Amazon S3

Now you can create a Script Canvas script that downloads the file.

To create a Script Canvas script to download a file from Amazon S3

1. In Lumberyard Editor, click Tools, Script Canvas.
2. In the Node Palette, expand Utilities and drag On Graph Start to the canvas.
3. On the right, in Variable Manager, click Create Variable.
4. In the Variable Type search box, enter AWSBehaviorS3Download, or scroll down to AWSBehaviorS3Download.
5. Click AWSBehaviorS3Download. In Node Inspector, AWSBehaviorS3Download Variable appears. The default name of the variable is Variable 1.
6. In Node Inspector, expand AWSBehaviorS3Download to show the input fields. Type the values in the following table into the corresponding boxes.

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BucketName</td>
<td>CloudGemAWSBehavior.s3nodeexamples</td>
</tr>
<tr>
<td>KeyName</td>
<td>downloadtest.txt</td>
</tr>
<tr>
<td>LocalFileName</td>
<td>s3downloadtest.txt</td>
</tr>
<tr>
<td>ContentType</td>
<td>text/html</td>
</tr>
</tbody>
</table>

7. From the Variable Manager, press Shift and drag Variable 1 AWSBehaviorS3Download to the canvas to create the Get Variable 1 node. (You can also drag AWSBehaviorS3Download to the canvas and then select Get Variable from the drop-down menu.)
8. Connect the Out pin of the On Graph Start node to the In pin of the Get Variable 1 node.
9. In the Node Palette, expand AWS, AWSBehaviorS3Download.
10. Drag the Download node to the right of the Get Variable 1 node on the canvas.
11. Connect the Out pin of the Get Variable 1 node to the In pin of the Download node.
12. Connect the AWSBehaviorS3Download pin of the Get Variable 1 node to the AWSBehaviorS3Download:0 pin of the Download node.

Step 3: Add Nodes to Display the Result

Next, to see the success or failure of the operation, you add AWSBehaviorS3DownloadNotificationBus nodes and a Print node to the graph. The nodes monitor for AWSBehaviorS3DownloadNotificationBus events and show you the result in the Lumberyard console window.

To display the success or failure of the download operation in the console window

1. In the Node Palette, expand AWS, AWSBehaviorS3DownloadNotificationBus.
2. Drag OnError to the canvas and place the node under the three nodes that are already connected.
3. Drag OnSuccess to the canvas and place it under the OnError node.
4. In the Node Palette, expand Utilities, Debug and drag Print to the right of the two AWSBehaviorS3DownloadNotificationBus nodes on the canvas. The Print node displays messages in the Lumberyard Editor console.
5. Connect the Out pins of the AWSBehaviorS3DownloadNotificationBus OnError and OnSuccess nodes to the In pin of the Print node.
6. Connect the String pins of the AWSBehaviorS3DownloadNotificationBus OnError and OnSuccess nodes to the Value pin on the Print node. Your canvas should look similar to the following image:
7. Save the canvas with a name like `MyS3DownloadTest.scriptcanvas`. The default file location is the `lumberyard_version\dev\project_name\scriptcanvas` directory.

8. Exit the Script Canvas editor.

Step 4: Test Download the File from Amazon S3

Now you are ready to attach the script canvas to an entity and test it.

To test downloading a file from Amazon S3

1. In Lumberyard Editor, right-click the viewport and choose Create entity.
2. In Entity Inspector, click Add Component.
3. Under Scripting, click Script Canvas to add a Script Canvas (p. 818) component to the entity.
4. Under Script Canvas, click the browse button (…).
5. In the Pick Script Canvas dialog box, choose the canvas that you created - for example, `mys3downloadtest (Script Canvas)`, and then click OK.
6. If the Lumberyard console window is not already open, press ~ or choose Tools, Console to open the console window. If the console window is already open and you want to clear it, press Ctrl+Shift+C.
7. Press **Ctrl+G** to start the level. The console reports the success of the operation as follows.

```
(Script Canvas) – File Downloaded
```

The file `s3downloadtest` file is downloaded to the `lumberyard_version\dev` directory.

### Generating an Amazon S3 Presigned URL With Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can use [presigned URLs](#) to share Amazon S3 objects with others. The **Presign** node in Script Canvas generates a presigned URL that links to an Amazon S3 object that you specify.

**Add Nodes that Generate a Presigned URL**

**To use the Presign node to presign a URL**

1. Upload a plain text file named `presigntest.txt` to the `s3nodeexamples` bucket.
2. In Lumberyard Editor, choose **Tools, Script Canvas**.
3. In the **Node Palette**, expand **Utilities** and drag **On Graph Start** to the canvas.
4. In **Variable Manager**, click **Create Variable**.
5. In the **Variable Type** search box, enter `AWSBehaviorS3Presign`, or scroll down to `AWSBehaviorS3Presign`.
6. Click `AWSBehaviorS3Presign`. In **Node Inspector**, `AWSBehaviorS3Presign Variable` appears. The default name of the variable is **Variable 1**.
7. In **Node Inspector**, expand **AWSBehaviorS3Presign** to show the input fields. Type or choose the values in the following table.

<table>
<thead>
<tr>
<th>Input Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BucketName</td>
<td>CloudGemAWSBehavior.s3nodeexamples</td>
</tr>
<tr>
<td>KeyName</td>
<td>presigntest.txt</td>
</tr>
<tr>
<td>RequestMethod</td>
<td>GET</td>
</tr>
<tr>
<td>ContentType</td>
<td>text/html</td>
</tr>
</tbody>
</table>

8. From the **Variable Manager**, press **Shift** and drag **Variable 1 AWSBehaviorS3Presign** to the canvas to create the **Get Variable 1** node. (You can also drag `AWSBehaviorS3Presign` to the canvas and then select **Get Variable** from the drop-down menu.)
9. Connect the **Out** pin of the **On Graph Start** node to the **In** pin of the **Get Variable 1** node.
10. In the **Node Palette**, expand **AWS, AWSBehaviorS3Presign**.
11. Drag the **Presign** node to the right of the **Get Variable 1** node on the canvas.
12. Connect the **Out** pin of the **Get Variable 1** node to the **In** pin of the **Presign** node.
13. Connect the `AWSBehaviorS3Presign` pin of the **Get Variable 1** node to the `AWSBehaviorS3Presign:0` pin of the **Presign** node.
Add Nodes to Display the Result

Next, to see the success or failure of the operation, you add `AWSBehaviorS3PresignNotificationBus` nodes and a `Print` node to the graph. The nodes monitor for `AWSBehaviorS3PresignNotificationBus` events and show you the result in the Lumberyard console window.

**To display the success or failure of the presign operation in the console window**

1. In the **Node Palette**, expand `AWS, AWSBehaviorS3PresignNotificationBus`.
2. Drag **OnError** to the canvas. Place the node under the three nodes that are already connected.
3. Drag **OnSuccess** to the canvas and place it under the **OnError** node.
4. In the **Node Palette**, expand **Utilities, Debug** and drag **Print** to the right of the two `AWSBehaviorS3PresignNotificationBus` nodes on the canvas. The **Print** node displays messages in the Lumberyard Editor console.
5. Connect the **Out** pins of the `AWSBehaviorS3PresignNotificationBus OnError` and **OnSuccess** nodes to the **In** pin of the **Print** node.
6. Connect the **String** pins of the `AWSBehaviorS3PresignNotificationBus OnError` and **OnSuccess** nodes to the **Value** pin on the **Print** node. Your canvas should look similar to the following image:
7. Save the canvas with a name like `MyS3PresignTest.scriptcanvas`. The default file location is the `lumberyard_version\dev\project_name\scriptcanvas` directory.

8. Exit the Script Canvas editor.

**Test Generate a Presigned URL**

Now you are ready to attach the script canvas to an entity and test it.

**To test the presigned Amazon S3 URL**

1. In Lumberyard Editor, right-click the viewport and choose `Create entity`.
2. In Entity Inspector, click `Add Component`.
3. Under Scripting, click `Script Canvas` to add a `Script Canvas (p. 818)` component to the entity.
4. Under Script Canvas, click the browse button (…).
5. In the Pick Script Canvas dialog box, choose the canvas that you created – for example, `mys3presigntest (Script Canvas)`, and then click OK.
6. If the Lumberyard console window is not already open, press ` or choose Tools, Console to open the console window. If the console window is already open and you want to clear it, press Ctrl+Shift+C.

7. Press Ctrl+G to start the level. The console reports the success of the operation, as in the following example.

```
 invaders.txt - https://s3.amazonaws.com//presigntest.txt?X-Amz-Algorithm=AWS4-HMAC-SHA256%26X-Amz-Credential={string}
```

### Invoking an AWS Lambda Function from Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **InvokeAWSLambda** node to invoke an AWS Lambda function.

**Note**

This tutorial uses an example Lambda function that is included with Lumberyard. For information on using your own Lambda function, see Making a Custom Lambda Function Available to Script Canvas (p. 2149).

**Topics**

- Prerequisites (p. 2146)
- Step 1: Add Nodes to Invoke an AWS Lambda Function (p. 2146)
- Step 2: Add Nodes to Display the Result (p. 2147)
- Making a Custom Lambda Function Available to Script Canvas (p. 2149)

**Prerequisites**

To complete the procedures in this tutorial, perform the following steps.

1. In the Project Configurator, choose the **CloudGemSamples** project.
2. Use Cloud Canvas Resource Manager or the `lmbr_aws` command to upload the resources for the project. This creates a project stack and a deployment in AWS.

After you perform these steps, your `lumberyard_version\dev\CloudGemSamples\Config\deployment_name.player.awsLogicalMappings.json` file should have an entry for CloudGemAWSScriptBehaviors.AWSBehaviorLambdaExample like the following example:

```
"CloudGemAWSScriptBehaviors.AWSBehaviorLambdaExample": {  
  "PhysicalResourceId": "CloudGemSamples-cgsamples-AWSBehaviorLambdaExample-T4F05C8V5QQR",  
  "ResourceType": "AWS::Lambda::Function"
},
```

### Step 1: Add Nodes to Invoke an AWS Lambda Function

The following procedure shows you how to create a Script Canvas graph that invokes an AWS Lambda function.

**To create a graph that invokes an AWS Lambda function**

1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the **Node Palette**, expand **Utilities** and drag the **On Graph Start** node to the graph.

3. On the right, in **Variable Manager**, click **Create Variable**.

4. In the **Variable Type** search box, enter **AWSLambda**, or scroll down to **AWSLambda**.

5. Click **AWSLambda**. In **Node Inspector**, **AWSLambda Variable** appears. The default name of the variable is **Variable 1**.

6. In **Node Inspector**, expand **AWSLambda** to show the **functionName** and **requestBody** boxes.

7. For **functionName**, select a Lambda function which is currently mapped by the Cloud Canvas Resource Manager. This tutorial uses the function **CloudGemAWSScriptBehaviors.AWSBehaviorLambdaExample**. You can find the example Lambda function in the `lumberyard_version\dev\Gems\CloudGemAWSScriptBehaviors\AWS\lambda-code\AWSBehaviorLambdaExample\main.py` file. The function simply returns a "Hello World" string.

8. For **AWSBehaviorLambdaExample**, leave **requestBody** blank. The **requestBody** field specifies a JSON string that can contain data to pass to the Lambda function.

9. From the **Variable Manager**, press **Shift** and drag **Variable 1 AWSLambda** to the graph to create the **Get Variable 1** node. (You can also drag **AWSLambda** to the graph and then select **Get Variable** from the drop-down menu.)

10. Connect the **Out** pin of the **On Graph Start** node to the **In** pin of the **Get Variable 1** node.

11. In the **Node Palette**, expand **AWS, AWSLambda**.

12. Drag the **InvokeAWSLambda** node to the right of the **Get Variable 1** node on the graph.

13. Connect the **Out** pin of the **Get Variable 1** node to the **In** pin of the **InvokeAWSLambda** node.

14. Connect the **AWSLambda** pin of the **Get Variable 1** node to the **AWSLambda:0** pin of the **InvokeAWSLambda** node.

**Step 2: Add Nodes to Display the Result**

Next, to see the success or failure of the operation, you add **AWSLambdaHandler** nodes and a **Print** node to the graph. The nodes monitor for **AWSLambdaHandler** events and show you the result in the Lumberyard console window.

**To display the output of the Lambda function in the console window**

1. In the **Node Palette**, expand **AWS, AWSLambdaHandler**.

2. Drag **OnError** to the graph. Place the node under the three nodes that are already connected.

3. Drag **OnSuccess** to the graph and place it under the **OnError** node.

4. In the **Node Palette**, expand **Utilities, Debug** and drag **Print** to the right of the two **AWSLambdaHandler** nodes on the graph. The **Print** node displays messages in the Lumberyard Editor console.

5. Connect the **Out** pins of the **AWSLambdaHandler OnError** and **OnSuccess** nodes to the **In** pin of the **Print** node.

6. Connect the **String** pins of the **AWSLambdaHandler OnError** and **OnSuccess** nodes to the **Value** pin on the **Print** node.

**Example**

Your graph should look similar to the following image.
7. Save the file with a name like `MyLambdaFunctionTest.scriptcanvas`. The default file location is the `lumberyard_version\dev\project_name\scriptcanvas` directory.

8. Close the Script Canvas editor.

**Step 3: Test the Lambda Function**

Now you are ready to attach the Script Canvas graph to an entity and test the script.

**To test the Lambda function**

1. In Lumberyard Editor, right-click the viewport and choose Create entity.
2. In the **Entity Inspector**, click **Add Component**.

3. Under **Scripting**, click **Script Canvas** to add a **Script Canvas** (p. 818) component to the entity.

4. Under **Script Canvas**, click the browse button (...).

5. In the **Pick Script Canvas** dialog box, choose the file that you created – for example, **mylambdafunctiontest (Script Canvas)**, and then click **OK**.

6. To open the Lumberyard console window, press `` or choose **Tools**, **Console**. If the console window is already open and you want to clear it, press **Ctrl+Shift+C**.

7. Press **Ctrl+G** to start the level. The console shows the "Hello World" result, as in the following example.

```python
general.enter_game_mode
Returned:
(Found resource management based identity pool %s.) - us-east-1:guid
(Found resource management based identity pool %s for authenticated access.) - us-east-1:guid
(CloudCanvas) - Anonymous Credentials pulled successfully for identity pool us-east-1:guid.
(Disable Accelerators)
(Script Canvas) - "Hello World"
```

### Making a Custom Lambda Function Available to Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com) or visit the [AWS Game Tech blog](https://aws.amazon.com) to learn more.

You can make custom Lambda function available to Script Canvas with the following steps:

1. Use the Project Configurator to add **Cloud Gem AWS Script Behaviors** to your project. For information on adding gems to your game project, see **Enabling Gems** (p. 1064).

2. In Lumberyard Editor, use the Cloud Canvas Resource Manager to add a Lambda resource to the **CloudGemAWSScriptBehaviors** resource group. When you perform this step, Lumberyard creates a directory named after your function and a placeholder **main.py** code file for you. The code directory and file look like the following:

   ```
lumberyard_version\dev\Gems\CloudGemAWSScriptBehaviors\AWS\lambda-code
your_function_name\main.py
```

3. Add your function code to the **main.py** file.

4. Use the Cloud Canvas Resource Manager to upload the resources to AWS. After you complete this step, your Lambda function becomes available to Script Canvas.

This topic shows you how to perform the second and fourth steps: use the Cloud Canvas Resource Manager to add a custom Lambda function to the **CloudGemAWSScriptBehaviors** resource group, and upload your Lambda function to AWS.

**Topics**

- Adding a Lambda Function Resource (p. 2150)
- Uploading Your Custom Lambda Function to AWS (p. 2152)
Adding a Lambda Function Resource

With Lumberyard Editor, you can add a Lambda function resource to the `CloudGemAWSScriptBehaviors` resource group.

To add a custom Lambda function to the CloudGemAWSScriptBehaviors resource group

1. In Lumberyard Editor, choose **AWS**, **Cloud Canvas**, **Resource Manager**.
2. Under **Resource Groups**, select **CloudGemAWSScriptBehaviors**.

3. Click **Add resource**, and then choose **Lambda function**.

![Resource Manager screenshot](image)
4. In the Add a Lambda function dialog box, specify the following values.

- **Name** – The name of the Lambda function.
- **Handler** – Specifies the code source for the function in the format `python_filename.function_name`. To use the `main.py` file that Lumberyard creates for you, specify `main.function_name`. For example, for a function named `MyCustomAWSLambdaFunction`, enter `main.MyCustomAWSLambdaFunction`.

For more information about Lambda function handlers, see Lambda Function Handler (Python) in the [AWS Lambda Developer Guide](#).

- **Player invokable** – Choose Yes. This option gives the player the permission to invoke the Lambda function.
These properties are added to the resource-template.json file in the
lumberyard_version\dev\Gems\CloudGemAWSScriptBehaviors\AWS directory.

5. Click OK. Now you are ready to add code to the main.py file that Lumberyard created for you at the following location:

lumberyard_version\dev\Gems\CloudGemAWSScriptBehaviors\AWS\lambda-code \your_function_name\main.py

Uploading Your Custom Lambda Function to AWS

After you add your Lambda function code to the main.py file, you are ready to upload it to AWS to make it available to Script Canvas.

To upload your Lambda function code to AWS

1. In Lumberyard Editor, choose AWS, Cloud Canvas, Resource Manager.
3. In Cloud Canvas Resource Manager, click Upload resources to upload your Lambda function to AWS.
4. In the Upload group resources dialog box, click Yes to approve any changes to security, if you agree.
Cloud Canvas Resource Manager notifies you when the Lambda resources have been created in AWS.

Now when you use Node Inspector to add an AWSLambda Variable node to a Script Canvas graph, your Lambda function becomes available in the functionName box.

For more information, see Invoking an AWS Lambda Function from Script Canvas (p. 2146).
Compute Farm Cloud Gem

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Compute Farm cloud gem is a tool for large-scale divide and conquer tasks that can be processed on a fleet of Amazon EC2 instances. This cloud gem is particularly suited for heavy-duty, computationally intensive Windows tasks like nav mesh generation, static lightmap baking, and terrain generation. In general, the Compute Farm cloud gem is designed for tasks that can be highly parallelized with recursive subdivision.

The Compute Farm cloud gem has the following three major aspects:

1. **Harness** – The brain of the cloud gem that orchestrates the task. The harness is a Python script that runs on every Amazon EC2 instance. The script continuously polls the Amazon Simple Workflow Service (SWF) for decisions that need to be made and activities that must be run. You can customize the harness for your own purposes.

2. **Amazon Machine Image (AMI)** – An image of the machine to do the task. The AMI must be prepared with all the necessary software to perform the task as well as a copy of the harness. The Compute Farm cloud gem includes a script to build the AMI.

3. **Launch Configuration and Auto Scaling Group** – After the AMI is created, you must create an Amazon EC2 Launch Configuration and an Amazon EC2 Auto Scaling Group to control the fleet of Amazon EC2 instances that will run the AMI. You set the Auto Scaling Group to the desired number of Amazon EC2 instances.

**Workflow Summary**

The harness receives and processes events from Amazon SWF, and runs three categories of task: divide, build and merge.

The harness must be extended with Python scripts that perform these three tasks for the target. In the dictionary sorting example that is included with the cloud gem, these tasks are the following:

- **Divide** a dictionary of words into multiple parts.
- **Build** a subset of the dictionary by sorting it.
- **Merge** the two sorted subsets back into one sorted subset.

For a high-level explanation and demonstration of the Compute Farm cloud gem for large scale terrain generation, see the 2018 GDC classroom session on YouTube. Developers who are interested in this particular implementation can reach out to us at lumberyard-feedback@amazon.com.

**Account Limits**

Your use of the Compute Farm cloud gem is subject to the limits on your AWS account. The following are the most relevant limits that you are likely to encounter:
• Limits on the number of on-demand Amazon EC2 instances that you can run for a particular type. For more information, see Q: How many instances can I run in Amazon EC2?

• Limits on your EBS volume storage. For more information, see Amazon Elastic Block Store (Amazon EBS) Limits in the Amazon Web Services General Reference.

• For large number of active instances, throttling limits on Amazon SWF. For more information, see Amazon SWF Limits in the Amazon Simple Workflow Service Developer Guide.

To request limit increases, see the AWS Support Center.

Defect Reporter Cloud Gem

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Defect Reporter cloud gem makes it easy for you and your developers and testers to gather, view, and manage defect reports from your game. The Defect Reporter cloud gem offers the following advantages:

• **Automated, error-free collection of data** – Instead of relying on the player to manually send the data, the most important data from the client is automatically captured. The contextual information that is included in the reports increases the likelihood of having actionable data and reduces the need to investigate with the customer.

• **Ease of reporting** – Players can submit reports from within the game without having to use email, forums, or open a browser. Players can easily report errors, which can increase the number of reports and reduce the number of unreported defects.

• **Ease of annotation** – Players can view the data that a report captures, annotate the report, and respond to any custom data collection queries that you provide.

Using Dynamic Content Manager

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Dynamic Content Cloud Gem and its Dynamic Content Manager to manage dynamic content updates for your game through AWS. You can use Dynamic Content Manager to create manifests and packages, add files to the packages, and upload the manifest and packages to the AWS Cloud. This ensures that your clients have the latest content for your game.

**Topics**

• Managing Dynamic Content Packages (p. 2165)

• Updating Dynamic Content (p. 2167)
- Using the Dynamic Content CLI (p. 2169)
- Testing the Dynamic Content System (p. 2177)
- Dynamic Content Engineering Details (p. 2181)

Prerequisites

This tutorial assumes the following:

- You are using a Lumberyard project that has the Dynamic Cloud Gem enabled (in the Project Configurator (p. 43), choose Cloud Gem Dynamic Content).
- You have created a project stack and deployment stack in Cloud Canvas Resource Manager (p. 2110) with the CloudGemDynamicContent resource group.

Note

If you do not have a deployment with the CloudGemDynamicContent resource group, the Dynamic Content Manager interface is disabled.

To use the Dynamic Content Manager to upload dynamic content

1. In Lumberyard Editor, choose AWS, Cloud Canvas, Dynamic Content Manager.

![Dynamic Content Manager](image)

2. In Dynamic Content Manager, click Create a new manifest.
A manifest is a file that keeps track of the dynamic content files that you deliver to customers. A manifest records the following information:

- The content files that have changed locally
- The files that are included in any file packages
- Differences between the local packages and the packages in the cloud that are staged for delivery to customers
- The target operating systems and devices for this manifest

You can create as many manifests as you require. For example, you can create specialized manifests for different types of assets.

3. Choose a name for the manifest. The name should reflect the purpose of the manifest.
4. Select the target operating systems and devices that you prefer to work on for this manifest.
5. When you create a manifest, the manifest is empty and has no files or packages.
Click **Add Files** or **Add Folder** and choose one of the target operating systems.

6. In the file browser, choose the files that you want to add to the manifest.
Because the original assets might not be in a form that your game can consume, the file browser opens to the location of your game projects asset cache. For example, your game OS might require textures to be in .dds format, but the original asset for a texture might be a .png file. The Asset Processor converts the files to the appropriate format and stores them in the asset cache, so the asset cache is your safest choice. However, if you have processed assets elsewhere in your file system, you are free to include them.

**File Warnings**

If the files that you selected are not supported by the current manifest or not supported by the OS that you selected, you are warned accordingly.

If the files that you selected are not in your game projects asset cache (and have therefore not been processed by the Asset Processor), you are notified of the error.

7. The left pane of Dynamic Content Manager shows the files that you added to the manifest. You can add files for different target operating systems and sort the files by their name, OS, or status. After you have some files in the file manager, you can select one or more files and remove them. You can remove the files by right-clicking them or by clicking Remove Files.
Now you are ready to add a package to the manifest.

Click **New Package**.

8. Enter a name for the package and choose the operating system or device type for it.

9. The new package appears in the right pane of **Dynamic Content Manager**. You can sort the packages by column.
Each package is stored in a .pak file, which is the archived file format that Lumberyard uses for its released game assets. You can have one or more packages per manifest, and you can assign the same file to multiple packages. This is useful when you have common files and operating system or device-specific files that you need to deliver in one package. It is also useful if you want to create bundles of items that might share assets.

10. To add files to the package that you created, drag the files from the manifest on the left to the package on the right. You can drag multiple files into the same .pak file.

You can drag operating system or device-specific files to a package with that OS type only. If you drag multiple files to a package and the OS type of these files does not match that of the package, Lumberyard warns you about the incompatible files.
The icons that appear in the **Packages** pane are described as follows.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>A file in a package has been updated in the asset cache. The contents of the package have been updated, so you might want to upload the package again.</td>
</tr>
<tr>
<td>✗</td>
<td>The package is not in the cloud. This can occur when the package has not yet been uploaded.</td>
</tr>
<tr>
<td>🔉</td>
<td>A package has changed with respect to its copy in the cloud. The contents of the package have been updated, so you might want to upload the package again.</td>
</tr>
<tr>
<td>✔️</td>
<td>The local package matches the package in the cloud. No action is necessary.</td>
</tr>
</tbody>
</table>

**Notes**

- You can remove individual files from a package, but to help prevent inadvertent removal, this action is limited to one file at a time.
- You can also use **Dynamic Content Manager** to delete packages, but deleting a package does not delete the staged packages in the cloud. To delete staged packages in the cloud, you must use the `lmbr_aws delete-uploaded-content` command to manage the service side of a cloud gem.

11. You can change the target operating systems or devices for the current manifest at any time.

   a. Click the settings button and choose **Change Target Platforms**.

   ![Dynamic Content Manager](image)

   b. Choose the operating systems or devices that you want. If none are selected, all supported operating systems and devices are added.
After you update the target operating systems and devices, **Dynamic Content Manager** lists only the files and packages for the ones that you selected. You can add files and folders only for the selected target OS or device.

12. When you are ready, click **Upload Packages**.

13. (Optional) To sign your packages, select **Sign packages with security key**.

14. If you haven't yet generated key pairs, click **Next** to generate a new key pair, and then click **Continue**.
You can also use the file menu to generate a new key pair at any time.

15. After the new key pair is generated, click **Upload**. This action both creates the package files locally and uploads them to your private staging bucket in the AWS Cloud.

The status bar of the main window shows upload process messages.
An animation in the **S3 Status** column indicates the package that is being uploaded.

By default, Lumberyard marks them as **Private**. When you are ready, you can transition the package to the **Public** stage to make it available for download by game clients. For more information on this step, see Package Stages (p. 2166).

### Managing Dynamic Content Packages

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Dynamic Content Cloud Gem provides an easy way for you to update your game content for your customers. After a release, you can update content without an app submission or traditional patch process. The Dynamic Content Cloud Gem provides a framework for you to group outdated assets into manifests. You can assign groups of assets together into .pak files (.zip files), and distribute these files to customers according to your own rules or schedules.

**Prerequisites**

To use the Dynamic Content Cloud Gem, you must meet these requirements:

- You must be using a Lumberyard Editor project that has the Dynamic Content Cloud Gem enabled (in the Project Configurator, select **Cloud Gem Dynamic Content**).
- You have a project stack created in Cloud Canvas Resource Manager (p. 2110).
- You have created a deployment stack in Cloud Canvas Resource Manager.
- You have created dynamic content packages that contain the updates. For more information, see Using Dynamic Content Manager (p. 2155).

To get a list of uploaded dynamic content packages, enter the following command

```bash
$ lmbr_aws dynamic-content list-uploaded-content
```

[WAF] Engine Root: C:\Amazon\Lumberyard\1.28.0.0\dev\WheelCustomization.manifest.pak PUBLIC {'Hash': 'af6282f222669294e2ac12ac59871e297', 'Size': '2048'}
Package Stages

You can use separate deployment stacks to handle different release workflows. For this purpose, the Dynamic Content Cloud Gem supports the following publishing stages.

- **Private** – The packages are never downloaded by the game client. When you upload new or altered content to the cloud from Dynamic Content Manager, your packages and manifest are always put in the Private stage. This empowers you to choose when and how your changes are released to customers.

- **Scheduled** – The packages are conditionally downloaded by the game client based on a specific date and time.

- **Public** – The packages are available to everyone that uses the deployment.

You can use these stages to add an additional layer of protection against inadvertent deployments of dynamic content to your players.

To transition a package from one stage to another, use the set-staging-status command to edit the package's staging status. See Editing Packages (p. 2166) later in this topic for an example.

Package Nesting

Each stage includes entries for your manifests and packages. Packages are nested according to the hierarchy that you determine when you create them. In the following example, the root-level manifest package is WheelCustomization.manifest.pak. It has two child asset packages, WheelCustomization_assets.shared.pak and WheelCustomization_data.shared.pak.

You can edit or delete root-level packages. However, you cannot edit child packages. You can only delete child packages.

Editing Packages

Use the set-staging-status command to edit the state of an uploaded root-level package. You can transition the content from one stage to another, schedule a window when it is available for download, select which child packages you would like to include in the transition, and much more. See the lmbr_aws dynamic-content set-staging-status (p. 2174) CLI documentation for full details.

Example: Transition a package to PUBLIC

This example shows how to transition a package and all its child packages to the public stage, using the filename of the root-level package.
Important

If you do not use `--include-children`, the child packages become orphans in the current stage and no longer visible to the game client.

Deleting Manifests and Packages

Use the `delete-uploaded-content` command to delete an uploaded manifest or package.

Example: Delete a manifest

This example shows how to delete a manifest, using the filename of the uploaded manifest. By including the `--confirm-deletion` flag, you confirm that you want to perform this deletion. Without this flag, the command will show you what would be deleted.

```
$ lmbr_aws dynamic-content delete-uploaded-content --file-path WheelCustomization.json --confirm-deletion
```

Tip

You can delete a specific version of a manifest using the optional `--version-id` argument.

Example: Delete a package and any children

This example shows how to delete a root package and all its child packages, using the filename of the uploaded root-level package. Without the `--confirm-deletion` flag, this command only shows what would be deleted.

```
$ lmbr_aws dynamic-content delete-uploaded-content --file-path WheelCustomization.manifest.pak
```

Tip

You can delete a specific version of a package using the optional `--version-id` argument.

Updating Dynamic Content

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To update dynamic content use the `lmbr_aws` CLI to upload new manifest content. We recommend that you enable dynamic content versioning to pre-stage and schedule the release of new versions so that players can still download previous versions while new content is uploaded. Dynamic content versioning also enables rollback to previous versions without having to re-upload the previous content.
Updating Dynamic Content with lmbr_aws

To perform dynamic content updates, enter the following command.

```
lmbr_aws dynamic-content upload-manifest-content --manifest-path <manifest name> --staging-status <PUBLIC|PRIVATE|WINDOW>
```

- `<manifest name>` – Specifies the filename of the manifest in the manifests directory (for example, `DynamicContentTest.json`).
- `<PUBLIC|PRIVATE|WINDOW>` – (Optional) Use PUBLIC to release the content immediately. Use WINDOW with `--start-date` and `--end-date` to specify a range during which the content will be publicly available. Defaults to PRIVATE.

Dynamic Content Versioning

Using dynamic content versioning, you can pre-stage and schedule the release of new dynamic content versions. Versioning your content is not required, but when enabled, provides the following advantages:

- Players can still download previous versions of your dynamic content while new content is uploaded, with no impact to availability. Without versioning enabled, clients will receive errors when packages in the S3 bucket are in the process of being updated, until the update completes.
- Developers have the ability to quickly rollback dynamic content to a previous version. Without versioning enabled, developers are required to re-upload previous content, which impacts availability and mitigation speed.

**Important**

When planning your versioning strategy, remember to take into consideration that normal S3 pricing applies to each version of an object.

Periodically, you might want to clear deprecated versions using the provided CLI commands, or define custom lifecycle management rules on the content bucket. However, please note that you will not be able to roll back to a version that has been cleared.

For more information on lifecycle management rules, see Object lifecycle management in the Amazon S3 Developer Guide.

Enabling dynamic content versioning

Dynamic content versioning is an optional feature. To use it, you must enable it by adding a tag to the deployment.

**To enable versioning on a new deployment**

Create the deployment with versioning enabled, as shown in the following CLI command:

```
lmbr_aws deployment create --deployment <deployment_name> --tags content-versioning
```

**To enable versioning on an existing deployment**

1. Add the `content-versioning` tag to the deployment.

   ```
   lmbr_aws deployment tags --add content-versioning --deployment <deployment_name>
   ```

2. Update the deployment stack.

   ```
   lmbr_aws deployment update --deployment <deployment_name>
   ```
Existing dynamic content table entries will also be migrated to the new `VersionedStagingSettingsTable` DynamoDB table after object versions are enabled.

Note that the dynamic content gem enables you to upload content with or without using manifests. When manifests are adopted, these files will also be versioned, and you have the option to update the staging status of all manifest content versions. However, if you skip manifests and upload a folder directly to the content bucket, you will have to manage the staging status of each `.pak` version separately.

**Suspending versioning**

To suspend dynamic content versioning, use the `suspend-versioning` CLI command. For details on the arguments available to this command, see dynamic-content suspend-versioning (p. 2175).

Note that suspending versioning will not revert the versioned S3 bucket to an unversioned state. Existing objects do not change after the suspension, but the S3 bucket will stop accruing new versions of the same object.

To resume versioning, follow the Enabling dynamic content versioning (p. 2168) instructions for an existing deployment.

**Using the Dynamic Content CLI**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you enable the Dynamic Content Cloud Gem, associated commands become available in the `lmbr_aws` CLI. You can then use these commands to update dynamic content and perform operations on manifest files and buckets.

The following `lmbr_aws` CLI extension commands are enabled by the Dynamic Content Cloud Gem.

**dynamic-content add-file-to-pak**

Add a given file to the specified `.pak` file.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--file-name <file_name>`
  
  File entry to add.

- `--manifest-path <manifest_path>`
  
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`. Defaults to `game\DynamicContent\Manifests\default.json`.

- `--pak-file <pak_file>`
  
  Pak file to add the file to.

- `--platform-type <OS_type>`
  
  (Optional) Operating system of the file entry to add.

**dynamic-content add-manifest-file**

Adds a file to the content manifest for the project.
In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--file-name <filename>**
  
  The name of the file, including the local directory (for example, staticdata/csv/gameproperties.csv, where local directory is staticdata/csv/.

- **--cache-root <cache directory>**

  (Optional) The reference for the local cache directory (for example, @assets@). The default is @assets@.

- **--bucket-prefix <prefix>**

  The bucket prefix under which to store the file in the content bucket.

- **--manifest-path <manifest_path>**

  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder game\DynamicContent\Manifests\.

  Defaults to game\DynamicContent\Manifests\default.json.

- **--output-root <default directory path>**

  (Optional) The path of the default directory to which to write. The default is @user@.

- **--platform-type <OS_type>**

  (Optional) The operating system type for the .pak file.

### dynamic-content add-pak

Add a new pak entry to the manifest.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--manifest-path <manifest_path>**

  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder game\DynamicContent\Manifests\.

  Defaults to game\DynamicContent\Manifests\default.json.

- **--pak-name <pak_name>**

  Name of the pak (the final filename will be <pak-name>..<OS>.pak).

- **--platform-type <OS_type>**

  (Optional) The operating system type for the .pak file.

### dynamic-content build-new-paks

Create .pak files based on manifest files which have changed.
In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--all`
  (Optional) Upload all .pak files regardless of the results of the file check.
- `--manifest-path <manifest_path>`
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`. Defaults to `game\DynamicContent\Manifests\default.json`.

**dynamic-content clear-dynamic-content**

Empty the bucket and table content.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

- `--all-versions`
  (Optional) Remove all versions of the manifest and pak files.
- `--noncurrent-versions`
  (Optional) Remove all the noncurrent versions of dynamic content. Requires confirmation on the command line following the use of this command, unless `--confirm-deleting-noncurrent-versions` is also used.
- `--confirm-deleting-noncurrent-versions`
  (Optional) Confirm that you know this command will delete all the noncurrent versions of files in the content bucket, and that you will not be able to roll back any previous version after this operation. Useful for automation.

**dynamic-content compare-bucket-content**

Compares manifest content to the bucket by checking HEAD metadata.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

- `--manifest-path <manifest_path>`
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`. Defaults to `game\DynamicContent\Manifests\default.json`.
- `--manifest-version-id <manifest version ID>`
  (Optional) Version of the standalone manifest pak. You can retrieve the available version ID's by using the `list-file-versions` command.

  Defaults to the latest version.

**dynamic-content create-new-manifest**

Create a new manifest.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--manifest-name <manifest_name>`
Name of the new manifest.

- **--manifest-path** `<manifest_path>`

(Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`. Defaults to `game\DynamicContent\Manifests\default.json`.

- **--target-platforms** `<target_OS> [<target_OS> ...]`

(Optional) Target operating systems for this new manifest (the default is all supported operating systems and devices).

### `dynamic-content delete-uploaded-content`

Delete a manifest or package.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--file-path** `<file_path>`

  The manifest or package file in the bucket.

- **--version-id** `<version_id>`

  (Optional) Version ID of the file. Defaults to the latest version.

- **--confirm-deletion**

  (Optional) Confirms that you want this command to delete the specified content. Omit this argument to preview what would be deleted.

### `dynamic-content generate-keys`

Generate a new public/private key pair for use by the dynamic content system.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

- **--key-name** `<key_name>`

  The name of the key file to use.

### `dynamic-content list-bucket-content`

Lists the manifest files in the content bucket.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

- **--manifest-path** `<manifest_path>`

  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`. Defaults to `game\DynamicContent\Manifests\default.json`.

- **--manifest-version-id** `<manifest version ID>`

  (Optional) Version of the standalone manifest pak. You can retrieve the available version ID's by using the `list-file-versions` command.
Defaults to the latest version.

**dynamic-content list-file-versions**

List all versions of a manifest or pak file found in the content bucket. Returns newest to oldest version based on uploaded date.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--file-name <filename>`
  (Optional) Name of the manifest or pak file.

**dynamic-content list-uploaded-content**

List all uploaded content.

The Common Arguments (p. 2391) can also be used with this command.

**dynamic-content migrate-staging-settings**

Migrate existing staging settings when content versioning is enabled or suspended. Existing data should be migrated automatically during the deployment update, so you will only need this command if the deployment fails to update.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--deployment-name <deployment name>`
  (Optional) Name of the deployment for which staging settings should be migrated.

  Defaults to the current active deployment.

**dynamic-content remove-manifest-file**

Removes a file from the content manifest for the project.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--file-name <filename>`

  Removes a file from the content manifest for the project. The file removed matches the value for `--file-name` that is specified in the `add-manifest-file` command.

- `--manifest-path <manifest_path>`

  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`.

  Defaults to `game\DynamicContent\Manifests\default.json`.

**dynamic-content request-url**

Requests a URL for the specified file.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

- `--file-path <file_path>`
The file in the bucket.

dynamic-content set-staging-status

Sets the staging status of the specified file.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--file-path <file_path>**
  The file in the bucket.
- **--version-id <version_id>**
  (Optional) Version ID of the file.
  Defaults to the latest version.
- **--staging-status <PUBLIC|PRIVATE|WINDOW>**
  Use PUBLIC to make the file immediately public. Use WINDOW with start-date and end-date to specify a range during which the file will be publicly available.
- **--start-date <start_date>**
  (Optional) Start date value for windowed staging. Use NOW or UTC date/time in the format "Jan 15 2021 14:30".
- **--end-date <end_date>**
  (Optional) End date value for windowed staging. Use NEVER or UTC date/time in the format "Jan 31 2021 14:30".
- **--include-children**
  (Optional) Set the staging status for all the children paks.
  Defaults to false.

**Warning**
There is a risk of a race condition if the staging status of a file is updated while a new version of the same file with a different staging status is uploaded at the same time from a different machine. The file could end up with an unexpected staging status in this case. To avoid this issue, specify the version ID when using the set-staging-status command.

dynamic-content show-logs

Show recent log events.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--minutes <minutes>**
  (Optional) The number of minutes of log events to show prior to now.
  Defaults to 10 minutes.

dynamic-content show-manifest

List all entries in the content manifest.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:
• --file-name <filename>
  (Optional) The file entry (local directory + key) to show.
• --manifest-path <manifest_path>
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a
  filename that exists in the folder game\DynamicContent\Manifests\. Defaults to game\DynamicContent\Manifests\default.json.
• --manifest-version-id <manifest version ID>
  (Optional) Version of the standalone manifest pak. You can retrieve the available version ID's by using
  the list-file-versions command. Defaults to the latest version.
• --platform-type <OS_type>
  (Optional) The operating system type of the file to list.
• --section <section>
  (Optional) Section to show (Paks or Files).

**dynamic-content show-signature**

Show the signature which would be created for a specified file.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

• --file-name <file_name>
  (Optional) The filename for which to show the signature.

**dynamic-content suspend-versioning**

Suspend dynamic content versioning.

Note that suspending versioning will not revert the versioned S3 bucket to an unversioned state. Existing
objects do not change after the suspension, but the S3 bucket will stop accruing new versions of the
same object.

To resume versioning, follow the instructions to Enabling dynamic content versioning (p. 2168) on an
existing deployment.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --deployment-name <deployment name>
  (Optional) Name of the deployment for which versioning should be suspended. Defaults to the current active deployment.
• --confirm-versioning-suspension
  (Optional) Confirms that you know this command will suspend content versioning, and that you need
to re-enable and update the deployment to resume versioning. Useful for automation.

**dynamic-content test-signature**

Tests whether a base64 signature is valid for the specified string.
In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--signature** `<signature>`  
  Base64 encoded signature.
- **--to-sign** `<string_to_sign>`  
  The string to sign.

**dynamic-content update-manifest**

Updates the manifest with current file hashes.

In addition to the Common Arguments (p. 2391), this command accepts the following argument:

- **--manifest-path** `<manifest_path>`  
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`.  
  Defaults to `game\DynamicContent\Manifests\default.json`.

**dynamic-content update-target-platforms**

Update the target operating system of a manifest file.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--manifest-path** `<manifest_path>`  
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`.  
  Defaults to `game\DynamicContent\Manifests\default.json`.
- **--target-platforms** `<target_OS> [<target_OS> ...]`  
  (Optional) Updated target operating systems for this new manifest (the default is all supported operating systems and devices).

**dynamic-content upload-manifest-content**

Updates the manifest and uploads changed manifest content to the content bucket.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- **--manifest-path** `<manifest_path>`  
  (Optional) Path and filename of the manifest file to use. You can specify either a full path or a filename that exists in the folder `game\DynamicContent\Manifests\`.  
  Defaults to `game\DynamicContent\Manifests\default.json`.
- **--deployment-name** `<deployment_name>`  
  (Optional) Which deployment to upload content to.  
  Defaults to the current active deployment.
- **--staging-status** `<PUBLIC|PRIVATE|WINDOW>`
(Optional) Use PUBLIC to make the new content immediately public. Use WINDOW with start-date and end-date to specify a range during which the content will be publicly available.

Defaults to PRIVATE.

- **--start-date** `<start_date>`
  (Optional) Start date value for windowed staging. Use NOW or UTC date/time in the format "Jan 15 2021 14:30".

- **--end-date** `<end_date>`
  (Optional) End date value for windowed staging. Use NEVER or UTC date/time in the format "Jan 31 2021 14:30".

- **--signing**
  (Optional) Add file signatures to the content table for client side verification.

  Defaults to no signature.

- **--invalidate-existing-files**
  (Optional) Invalidate existing files with the same name in the CloudFront edge cache. Only effective if dynamic content is delivered via CloudFront. For details on using this service with dynamic content, see the topic on Using Amazon CloudFront (p. 2185).

  Defaults to no invalidation.

- **--replace**
  (Optional) Removes older versions when a new version has been uploaded.

- **--all**
  (Optional) Updates all content in the manifest regardless of whether it appears to have been updated already.

### Testing the Dynamic Content System

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard contains a basic level called **DynamicContentTest** that you can use to verify that the dynamic content system is functioning properly. The level is located in the `dev\CloudGemSamples\Levels\CloudGemTests\DynamicContentTest` directory.

**Prerequisites**

To test any cloud gem, you must have a deployment with the necessary resources. These resources are created through the Cloud Canvas Resource Manager. For detailed instructions on creating these resources, see Initializing Cloud Canvas Resource Manager (p. 2349).

For the test, you are likely to play the role of both content creator and game client. Because the editor prefers loose files to .pak files, you might want to run the engine in game client mode instead of editor mode. When you run the engine in editor mode, updated data does not take precedence during run time. If you do want to run the test in editor mode, you can change this setting by entering the console command `sys_PakPriority=1` before you start the test. After testing, if you want the editor to return to preferring loose files (the default), enter the console command `sys_PakPriority=0`.

Version 1.28
2177
To run the level as a game client, you must export the mappings from your deployment stack to your `\dev\project_name\Config` directory. As of Lumberyard 1.11, separate mappings are exported for the player game client and game server to differentiate the resources that are visible to each. To export the mappings, select your deployment in Cloud Canvas Resource Manager and click Export Mapping. The exported files have the format `<deployment_name>.player.awsLogicalMappings.json` and `<deployment_name>.server.awsLogicalMappings.json`.

After you have exported a mapping to the Config directory, you can use the mapping when you run a game launcher such as `\dev\Bin64vcNNN\CloudGemSamplesLauncher.exe`. If you have more than one mapping, use the `-cc_override_resource_map <mappings_file>` option to specify one when you run the launcher. For more information, see Selecting a Deployment with a PC Launcher (p. 2358).

The test level contains a button that requests an update of a predefined dynamic content manifest that describes a single `.pak` file. The `.pak` file references a `gameproperties.csv` file. The static data system uses the properties file to reflect a property value to Lua and to a text control that displays the property value.

When the level starts, the text box displays the current value of the property. Pressing the button starts the update process. During the update, the client checks the dynamic content system for new data. When it detects new data, it downloads the data and reflects the new value.

### Data Preparation

To create data that is considered "new" for the purposes of the test, perform the following steps.

#### To prepare data for the test

1. Open the `gameproperties.csv` file in the `\dev\CloudGemSamples\StaticData\CSV` directory. The file contents are similar to the following:

   ```
   GameProperty,Value
   DynamicMessage,"LOCAL DATA"
   ```

2. Set the "LOCAL DATA" message to a string that you want to receive when the data is updated (for example, "UPDATED").

3. Save the file.

4. Do one of the following:
   - At a command line prompt, enter the following command:
     ```
     lmbr_aws dynamic-content upload-manifest-content --manifest-path DynamicContentTest.json --staging-status PUBLIC
     ```
   - Use the packer tool to upload the manifest content. The manifest file that packs up `gameproperties.csv` is `DynamicContentTest.json`. Normally you should not have to alter `DynamicContentTest.json` for testing purposes. However, if you want to use it to ensure that your test contains no old test data, configure it like this:

   ```json
   {
     "Files": [
       {
         "bucketPrefix": "",
         "cacheRoot": "@assets@",
         "hash": "",
         "keyName": "gameproperties.csv",
         "localFolder": "StaticData/CSV",
     }
   }
   ```
At this point, you have created and uploaded a .pak file that contains the current version of your gameproperties.csv file and your manifest.

5. Stage the uploads as PUBLIC, which means that they are ready for download. If you used the command line option in the previous step to upload your data, this was done for you with the --staging-status PUBLIC option.

   If you used the packer tool to perform the upload, transition the package to PUBLIC using the following command:

   ```
   lmbr_aws dynamic-content set-staging-status --file-path DynamicContentTest.json --staging-status PUBLIC
   ```

6. Change your gameproperties.csv file message back to "LOCAL DATA" or its original message. Do this so that you can be sure that your updated message comes from the downloaded content.

Running the Test

After you have saved the gameproperties.csv file to reflect local data, you can run the test.

To test the dynamic content system

1. Start the DynamicContentTest level. When you start the level, you should see the most recent message that you entered in the game properties file ("LOCAL DATA" in step 6 of Data Preparation (p. 2178)).

2. Click Update. You should see the update in 3 to 10 seconds, depending on Lambda function latency.

Diagnosing Failures

When diagnosing a failure, you must determine how far your updated data went in the processing chain.

Check the .pak File for Upload

First, open the .pak file that was to be uploaded with 7-Zip or a similar utility. The file is in the location CloudGemSamples\DynamicContent\Paks\DynamicContentTest.shared.pak. The .pak file contains the .csv file that the end client should have received. If something is wrong with the .pak file, the packing process was faulty and should be repeated. If so, delete the .pak file. In your gameproperties.csv file, edit the message you want. Configure the DynamicContentManifest.json as indicated in the Data Preparation (p. 2178) section earlier in this document. Run the upload command again. Finally, recheck that your .pak file has the correct .csv file and data.

Check the Downloaded .pak File

If your local .pak file for upload is free of errors, check the file that was downloaded. The downloaded file is located in the directory for your game project (for example, CloudGemSamples\pc\user\dynamiccontent\paks\DynamicContentTest.shared.pak). Perform the same inspection on this file as you did for the file in the upload location. If the data is what you expect, the client did not receive the new data. This can occur because of a download failure, or because the data in the Amazon
S3 bucket is incorrect. You can use the AWS console to inspect the corresponding files in your Amazon S3 bucket.

If the downloaded .pak file was updated correctly, then the problem might be one of the following:

- The dynamic content system itself did not reload the file.
- The dynamic content system did not accept the new data. Check whether $sys_PakPriority$ is set to 0. In editor mode, you must set this to 1. For more information, see the Prerequisites (p. 2177) section earlier in this document.
- The test is not reflecting the file correctly and requires further diagnosis.

**Test Implementation Details**

Because DynamicContent and StaticData are system components, they do not have to be added to the entity in the test level. Since in most cases the dynamic content that Lumberyard loads would persist longer than any entity, the example here shows you how to use the SystemComponent. The entity in this level contains only Lua script and test logic.

1. In the script, the example connects to the EBus, loads the canvas, and waits for the button event.

   ```lua
   function DynamicContentTest:OnAction(entityId, actionName)
       Debug.Log("Received Action: ", actionName)

       if actionName == "ButtonPressed" then
           if DynamicContentRequestBus.Event == nil then
               Debug.Log("No Content Request Events found")
           return
           end
           DynamicContentRequestBus.Event.RequestManifest(SystemEntityId, "DynamicContentTest.json")
           self:UpdateText()
       end
   end
   ```

   When the button is pressed, the code requests the new manifest. The content specified by the manifest is automatically retrieved.
2. After the content is retrieved, a `NewPakContentReady` event fires.

```javascript
function DynamicContentTest:NewPakContentReady(outputFile)
    Debug.Log("Received pak update: " .. outputFile)
    self:UpdateGameProperties()
end
```

In the test example, you already know that the `.pak` file contains game properties data. In more complex implementations, you might need to query the contents of the file.

3. The `UpdateGameProperties()` function requests that the static data system load the `.csv` file.

```javascript
function DynamicContentTest:UpdateGameProperties()
    if StaticDataRequestBus:HasEventAdded(StaticDataRequestBus::Event_Nil)
        Debug.Log("No StaticData Request Events Found")
        return
    end
    StaticDataRequestBus:RequestAddEventForStaticData(StaticDataRequestBus::Event_LoadRelativeFile(SystemEntityId, "StaticData/CSV/gameproperties.csv")
end
```

4. The file load triggers the `TypeReloaded` event, which calls the `UpdateText()` function.

```javascript
function DynamicContentTest:TypeReloaded(outputFile)
    if outputFile == "gameproperties" then
        self:UpdateText()
    end
end
```

At this point, the user interface text has been updated with the new game property value.

5. Because the example attached to the `SystemComponent`, and the sample might be running in the editor, the cleanup routine manually removes the dynamic content.

```javascript
function DynamicContentTest:OnDeactivate()
    DynamicContentRequestBus:ClearAllContent(SystemEntityId)
    self.uiCanvasNotificationLuaBusHandler:Disconnect()
    self.dynamicContentUpdateBus:Disconnect()
    self.staticDataUpdateBus:Disconnect()
end
```

### Dynamic Content Engineering Details

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This topic provides programmatic details about the dynamic content update process. This includes manifest file information, Dynamic Content Cloud Gem EBus events, and Dynamic Content Cloud Gem service API. For information about the `lmbr_aws` CLI extensions enabled by the Dynamic Content Cloud Gem, see Using the Dynamic Content CLI (p. 2169).

#### Manifest File

In your Lumberyard installation, the default location of the manifest file is `<GameFolder>/AWS/DynamicContent/DynamicContentManifest.json`.

The following is a simple example manifest for the `SamplesProject` `DontDie` sample.
"Files": [ 
{ 
"hash": "3bebdb5bd88c79c42e5f7f3dc4e900", 
"outputRoot": "@user@", 
"bucketPrefix": "static-data", 
"keyName": "gameproperties.csv", 
"cacheRoot": "@assets@", 
"platformType": ", 
"localFolder": "staticdata/csv" 
} 
] 

The following table describes the properties in the manifest file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash</td>
<td>MD5 hash of the file.</td>
</tr>
<tr>
<td>outputRoot</td>
<td>Base output directory.</td>
</tr>
<tr>
<td>bucketPrefix</td>
<td>Prefix inside the bucket for the file</td>
</tr>
<tr>
<td>keyName</td>
<td>Name of the file key in the bucket, which will be appended to the beginning of the hash. The final key name has the format bucketPrefix/keyName.</td>
</tr>
<tr>
<td>cacheRoot</td>
<td>Root directory to search for copies of the outdated file asset.</td>
</tr>
<tr>
<td>platformType</td>
<td>Windows (pc), macOS (osx_gl), or Linux (linux). An empty value specifies all operating systems.</td>
</tr>
<tr>
<td>localFolder</td>
<td>Directory to write locally within the outputRoot. The full output has the format outputRoot/localFolder/keyName.</td>
</tr>
</tbody>
</table>

**EBus Events**

The Dynamic Content Cloud Gem provides an EBus API and includes calls exposed to Lua. The basic top-level update request looks like this:

```lua
CloudCanvas::DynamicContent::DynamicContentRequestBus::BroadcastResult(
    requestSuccess, 
    &CloudCanvas::DynamicContent::DynamicContentRequestBus::Events::RequestManifest, 
    manifestName)
```

requestSuccess (bool) – Specifies whether the request was successfully sent.

manifestName (char*) – Name of the manifest file (for example, DynamicContentTest.json). The system handles .pak file and operating system naming conventions (for example, DynamicContentTest.shared.pak).

**Requesting pak files without using manifest**

Use the following API calls to request pak files without using a manifest:

```lua
CloudCanvas::DynamicContent::DynamicContentRequestBus::BroadcastResult(
    requestSuccess, 
    &CloudCanvas::DynamicContent::DynamicContentRequestBus::Events::UpdateFileStatusList, 
    uploadRequests,
```
autoDownload);

requestSuccess (bool) – Specifies whether the request was successfully sent.

uploadRequests (AZStd::vector<AZStd::string>) – List of bucket keys.

autoDownload (bool) – Specify true to download files automatically.

CloudCanvas::DynamicContent::DynamicContentRequestBus::BroadcastResult(
    requestSuccess,
    &CloudCanvas::DynamicContent::DynamicContentRequestBus::Events::UpdateFileStatus,
    fileName,
    autoDownload);

requestSuccess (bool) – Specifies whether the request was successfully sent.

fileName (char*) – Name of the pak file to request.

autoDownload (bool) – Specify true to download files automatically.

Versioning Support

Requesting pak files using versioned manifest

Use the following API calls to request pak files using a manifest when versioning is enabled:

CloudCanvas::DynamicContent::DynamicContentRequestBus::BroadcastResult(
    requestSuccess,
    &CloudCanvas::DynamicContent::DynamicContentRequestBus::Events::RequestVersionedManifest,
    manifestName,
    versionId);

requestSuccess (bool) – Specifies whether the request was successfully sent.

manifestName (char*) – Name of the manifest file (for example, DynamicContentTest.json).

The system handles .pak file and operating system naming conventions (for example, DynamicContentTest.shared.pak).

versionId (char*) – Version ID of the manifest. Uses the current active (public) version if not specified.

CloudCanvas::DynamicContent::DynamicContentRequestBus::BroadcastResult(
    requestSuccess,
    &CloudCanvas::DynamicContent::DynamicContentRequestBus::Events::RequestVersionedFileStatus,
    fileName,
    outputFile,
    versionId);

requestSuccess (bool) – Specifies whether the request was successfully sent.

fileName (char*) – Name of the pak file to request.

outputFile (char*) – Name of the output pak file.

versionId (char*) – Version ID of the file. Uses the current active (public) version if not specified.

Requesting pak files without using manifest

Use the following API calls to request files without using a manifest when versioning is enabled:
requestSuccess (bool) – Specifies whether the request was successfully sent.

requestMap (AZStd::unordered_map<AZStd::string, AZStd::string>) – Map of file names to file version IDs.

autoDownload (bool) – Specify true to download files automatically.

requestSuccess (bool) – Specifies whether the request was successfully sent.

fileName (char*) – Name of the pak file to request.

autoDownload (bool) – Specify true to download files automatically.

versionId (char*) – Version ID of the file. Uses the current active (public) version if not specified.

Manifest Received
The following EBus events are triggered when a manifest has been received successfully or unsuccessfully.

Success

EBUS_EVENT(CloudCanvas::DynamicContent::DynamicContentRequestBus, ManifestUpdated, bucketName, bucketPrefix)

When all .pak files are complete, a RequestCompleted event is broadcast.

Failure

EBUS_EVENT(CloudCanvas::DynamicContent::DynamicContentRequestBus, ManifestFailed, bucketName, bucketPrefix, errorStr)

Service API
The Dynamic Content Cloud Gem exposes API calls through Amazon API Gateway for the game client.

The following table lists the calls.

<table>
<thead>
<tr>
<th>Client API Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/client/content POST</td>
<td>Request presigned URLs for a list of files, based on the provided version IDs. The active version will be returned if no version ID is specified. Returns the URLs or a failure message.</td>
</tr>
</tbody>
</table>
Using Amazon CloudFront

Amazon CloudFront is a fast content delivery network (CDN) service that can extend S3 to securely deliver data to customers with low latency and high transfer speeds at an additional cost. You can take advantage of it when using the DynamicContent gem. To learn more about Amazon CloudFront, read the Amazon CloudFront documentation.

Enable the CloudFront extension

To enable the Amazon CloudFront feature, add deployment tag `content-distribution` when you create a deployment with CloudGemDynamicContent enabled:

```bash
imbr_aws deployment create --deployment {deployment_name} --tags content-distribution
```

To learn how to create a project stack and deployment stack using CloudCanvas command line, read Using the Cloud Canvas Command Line.

Using this deployment tag will add a few more AWS resources to your deployment stack including a Amazon CloudFront distribution and an S3 bucket. All of the signed URLs from the DynamicContent gem will be created using Amazon CloudFront automatically after the feature is enabled.

Create and upload CloudFront key pairs

Each AWS account that you use to create Amazon CloudFront signed URLs or signed cookies—your trusted signers—must have its own Amazon CloudFront key pair, and the key pair must be active. This is required for using Amazon CloudFront with the DynamicContent gem. Read the AWS document Creating Amazon CloudFront Key Pairs for Your Trusted Signers to learn how to create your own key pairs.

Note that IAM users can't create Amazon CloudFront key pairs. You must log in using root credentials to create key pairs. To learn more about the root credentials, read The AWS Account Root User.

After you have created and downloaded the key pairs, upload your private key using the following CLI command:

```bash
imbr_aws dynamic-content upload-cf-key --key-path {path_to_private_key} --deployment-name {deployment_to_use}
```

The expected name format for your private key is `pk-<accountkey>.pem`. Your key will be securely stored in an S3 bucket called `AccessBucket`.

Invalidate files from CloudFront edge caches

By default, Amazon CloudFront caches a response from Amazon S3 for 24 hours. If your request lands at an edge location that served the Amazon S3 response within 24 hours, Amazon CloudFront uses the cached response even if you updated the content in Amazon S3. In this case, you may get outdated content after uploading the same named files.

When updating your content you can invalidate the cache for files being update by adding the `--invalidate-existing-files` argument to your CLI command:

```bash
imbr_aws dynamic-content upload-manifest-content --manifest-path {path_to_manifest} --deployment-name {deployment_to_use} --staging-status {staging_status} --invalidate-existing-files
```

You can also invalidate the cached response for a specific file using the following CLI command:

```bash
imbr_aws dynamic-content invalidate-file --file-path {file_name_in_s3_bucket} --caller-reference {unique_identity}
```

`--caller-reference` requires a value that you specify to uniquely identify an invalidation request. Amazon CloudFront uses the value to prevent you from accidentally resubmitting an identical request. If you make a second invalidation request with the same value for caller reference, and if the rest of the request is the same, Amazon CloudFront doesn't create a new invalidation request.
Game Metrics Cloud Gem

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Game Metrics cloud gem to collect event data on player behaviors, which you can then analyze or trigger event actions. This gem uses a service API with the Cloud Gem Framework for event submission, Amazon S3 for data storage, Amazon Athena as the query engine, and Amazon QuickSight as the data visualizer.

You can enable this gem to do the following:

- View realtime data about users playing the game.
- Run queries and generate reports on player behavior over time.
- Visualize player activity in the game world.
- Record player events, even if players are offline.

**Note**
The Game Metrics cloud gem is only supported in regions where FIFO is supported. For more information, see Amazon SQS FIFO (First-In-First-Out) Queues in the Amazon Simple Queue Service Developer Guide.

Example Overview

The following illustrates how AWS services process game events.

When events are generated from your game, the following occurs:

1. Events submitted by REST requests are first processed by the Amazon API Gateway Lambda handler called the `FIFO_producer` (producer). This producer breaks the request into smaller chunks in order to upload a maximum of 256KB per chunk, which is the maximum payload size an SQS message can support. Because the size of a single event, including attributes, can't exceed 256KB, the producer returns an error if an event exceeds this limit.
2. A Lambda process called the **FIFO_consumer** (consumer) processes the SQS messages.

3. The consumer is triggered every five minutes and self-replicates when one of the following conditions are met:
   - SQS queue growth exceeds 5%, which is checked every 15 seconds.
   - SQS queue count exceeds 3000 messages.

The consumer processes as many SQS messages as possible within its five minute lifespan, aggregates the events of the same type into a single parquet file, and then saves the file to a specific S3 key.

4. Another Lambda function called the **amoeba_generator** is triggered every 20 minutes and aggregates all files in a single key path into a single file. This process maximizes IO (input/output) performance when loading the files by the database. AWS Glue crawlers crawl the data bucket and identify a common schema; this schema defines the tables for Athena to use.

5. The S3 files are made available to Athena for querying through AWS Glue. AWS Glue updates Athena's database schema every hour. The S3 event data is partitioned (or indexed) by the hour. This means the lowest level of granularity when using Athena partitions is by the hour.

   **Note**
   Sending queries based on partitions can greatly reduce costs and improve query performance. For more information, see Partitioning Data in the Amazon Athena User Guide.

The Game Metrics cloud gem includes the following default Athena partitions:

<table>
<thead>
<tr>
<th>Partitions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_event_name</td>
<td>Name of the event.</td>
</tr>
<tr>
<td>p_server_timestamp_strftime</td>
<td>Server time stamp (UTC) of the event in the format %Y%m%d%H0000.</td>
</tr>
<tr>
<td>p_server_timestamp_year</td>
<td>Year of the server time stamp (UTC).</td>
</tr>
<tr>
<td>p_server_timestamp_month</td>
<td>Month of the server time stamp (UTC).</td>
</tr>
<tr>
<td>p_server_timestamp_day</td>
<td>Day of the server time stamp (UTC).</td>
</tr>
<tr>
<td>p_server_timestamp_hour</td>
<td>Hour of the server time stamp (UTC).</td>
</tr>
<tr>
<td>p_event_source</td>
<td>Data source of the event. The default value is &quot;cloudgemmetric&quot; or &quot;cloudgemdefectreporter&quot;.</td>
</tr>
<tr>
<td>p_client_build_identifier</td>
<td>Build identifier associated with the event.</td>
</tr>
<tr>
<td>p_data_sensitivity</td>
<td>Encryption level of the data.</td>
</tr>
<tr>
<td>p_event_schema_hash</td>
<td>Hash of the event schema.</td>
</tr>
</tbody>
</table>

**In-Game Survey Cloud Gem**

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the In-Game Survey Cloud Gem to create surveys for your game and test them in the InGameSurveySample. Your players can see active surveys and submit answers to them.

**Leaderboard Cloud Gem**

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

You can use the Leaderboard cloud gem to create, view, and customize leaderboards by deleting scores, banning players, and removing bans.

**Message Of The Day Cloud Gem**

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

You can use the Message of the Day Cloud Gem to schedule messages that your game consumes.

**Player Account Cloud Gem**

The Player Account Cloud Gem provides a standalone player authentication and management solution that uses Amazon Cognito. The cloud gem includes a player registration API, built-in two-step verification, and support for both anonymous and authenticated players. You can use the `lmbr_aws` CLI to administer players and manually register, delete, ban, and update their accounts.

**Legal Restrictions**

You are responsible for (a) providing legally adequate privacy notices to your end users; (b) obtaining any necessary consent from the end user for the collection, use, transfer, and storage of any name, password,
other login information, or personally identifiable information or personal data of any end user that you (or any third-party plug-in or service provider you use) may access; (c) using and authorizing others to access and use the information only for the purposes permitted by the end user; and (d) ensuring the information is collected, used, transferred, and stored in accordance with all laws, rules, and regulations applicable in jurisdictions in which your applications are used.

Using the Player Account Cloud Gem

Player accounts have the following automatic or required fields:

- **Amazon Cognito Identity ID** – The unique ID of the player in the Amazon Cognito database. This ID field is initially empty and is filled after the player signs in for the first time.
- **Account ID** – An automatically assigned ID.
- **User Name** – The Amazon Cognito user name, required during account creation.
- **Email** – The email of the player, required during account creation.
- **Account Status** – The current status of the account. The following statuses are possible:
  - **Archived** – The account has been archived due to inactivity. You can change this from the Amazon Cognito console.
  - **Compromised** – This account requires further investigation from Amazon Cognito console.
  - **Disabled** – The player account is currently disabled. You can reenable it on the Amazon Cognito console.
  - **Force Change Password** – An AWS administrator has used Amazon Cognito to create the player account, and the player now must change their temporary password before they can successfully sign in.
  - **Reset Required** – The player's account was imported, but the player has not logged in.
  - **Unconfirmed** – The player account hasn't confirmed the account's email address or phone number. The account is unusable until it has been confirmed.
  - **Unknown** – The account is in an unknown state. Use the Amazon Cognito console to investigate further.

In addition, this gem supports setting the following optional fields:

- **Player Name**
- **Family Name**
- **Given Name**
- **Nickname**
- **Gender**
- **Locale**

Only the **Username** and **Email** fields are required when creating a new player account. Validation is performed server-side using the OpenID Connect specification rules.

Use the `lmbr_aws player-account` (p. 2189) CLI commands to add, edit, confirm, ban, and show players in your game.

Using the Player Account CLI

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
When you enable the Player Account Cloud Gem, associated commands become available in the `lmbr_aws` CLI. You can then use these commands to perform player account operations.

The following `lmbr_aws` CLI extension commands are enabled by the Player Account Cloud Gem.

**player-account add-player**

Create a new player account in Amazon Cognito. Validation is performed server-side using the OpenID Connect specification rules.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--username <username>`
  Amazon Cognito username of the account to create.
- `--email <email_address>`
  Email address of the player.
- `--playername <player_name>`
  (Optional) Player name of the player.
- `--familyname <family_name>`
  (Optional) Family name of the player.
- `--givenname <given_name>`
  (Optional) Given name of the player.
- `--nickname <nickname>`
  (Optional) Nickname of the player.
- `--gender <gender>`
  (Optional) Gender of the player. The values "male" and "female" are defined by the OpenID Connect specification, but any value may be used.
- `--locale <locale>`
  (Optional) Locale of the player.

**player-account ban-player**

Ban a player account by account ID.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--account-id <account_id>`
  Account ID to ban.

**player-account confirm-player**

Confirm a player account whose status is "Unconfirmed" to make it usable.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--username <username>`
Amazon Cognito username of the account to confirm.

**player-account edit-player**

Edit the attribute values of a player account by account ID. Validation is performed server-side using the OpenID Connect specification rules.

In addition to the [Common Arguments](p. 2391), this command accepts the following arguments:

- **--account-id** `<account_id>`
  
  Account ID to edit.

- **--email** `<email_address>`
  
  (Optional) New email address for the player.

- **--playername** `<player_name>`
  
  (Optional) New player name for the player.

- **--familyname** `<family_name>`
  
  (Optional) New family name for the player.

- **--givenname** `<given_name>`
  
  (Optional) New given name for the player.

- **--nickname** `<nickname>`
  
  (Optional) New nickname for the player.

- **--gender** `<gender>`
  
  (Optional) New gender for the player. The values "male" and "female" are defined by the OpenID Connect specification, but any value may be used.

- **--locale** `<locale>`
  
  (Optional) New locale for the player.

**player-account remove-player-ban**

Remove the ban on a player account by account ID.

In addition to the [Common Arguments](p. 2391), this command accepts the following arguments:

- **--account-id** `<account_id>`
  
  Account ID for which the ban should be removed.

**player-account reset-player-password**

Reset the password for a player account by username.

In addition to the [Common Arguments](p. 2391), this command accepts the following arguments:

- **--username** `<username>`
  
  Amazon Cognito username of the account for which the password should be reset.
player-account show-banned-players

List all banned players.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --page-token <page_token>
  (Optional) The pagination token to get to the next page.

player-account show-logs

Show recent log events.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --minutes <minutes>
  (Optional) Display log events for the specified number of minutes before now. The default is 10 minutes.

player-account show-players

List all registered players, using optional filters.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --filter-type <AccountId | PlayerName | CognitoIdentity | CognitoUsername | CognitoEmail>
  (Optional) Type of filter to apply.
• --filter-value <filter_value>
  (Optional) String value to use for the filter.
• --page-token <page_token>
  (Optional) The pagination token to get to the next page.

player-account show-player-details

List account details for a player by account ID.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --account-id <account_id>
  Account ID for which to show details.

Testing the Player Account Cloud Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To test the Player Account Cloud Gem, you can use the CloudGemSamples project that is included with Lumberyard. This sample contains a level named PlayerAccountSample. The player account sample
level provides a menu from which players can create an account, verify their email addresses, sign in and out of their accounts, and recover their passwords by email.

The Player Account Sample Level

The following table describes the files for the player account sample level.

<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\dev\CloudGemSamples \Levels \PlayerAccountSample</td>
<td>A level that contains an entity with the Lua script component.</td>
</tr>
<tr>
<td>\dev\CloudGemSamples \Scripts\PlayerAccount</td>
<td>The Lua scripts that manage the UI canvases and interact with the Player Account Cloud Gem's EBus API.</td>
</tr>
<tr>
<td>\dev\CloudGemSamples\UI \Canvases\PlayerAccount</td>
<td>The UI canvases displayed for the player.</td>
</tr>
</tbody>
</table>

Note that in the sample, UI canvases do not include functionality for displaying error messages graphically; instead, error messages are output to the console. If you integrate this gem into an existing project, you should use the UI canvas to show error messages. For example, you could do this when the confirmation code is incorrect or the password does not meet the minimum password length. For more information on using the Player Account Cloud Gem in an existing project, see Integrating the Player Account Cloud Gem into an Existing Project (p. 2196).

The following diagram shows how the UI canvases are connected. For more information on the Amazon Cognito user pool signup process, see the Signing Up and Confirming User Accounts page in the Amazon Cognito Developer Guide.

Prerequisites

This tutorial assumes the following:

- You are using a Lumberyard project that has **Cloud Gem Player Account** enabled.
- You have created a project stack in Cloud Canvas Resource Manager (p. 2110).
- You have a deployment stack with Player Account Cloud Gem resources in Cloud Canvas Resource Manager.
If you don’t meet the prerequisites, follow the steps in Enabling Gems (p. 1064) to add Cloud Gem Player Account in the Project Configurator. For information on creating a project stack and a deployment stack, see Tutorial: Getting Started with Cloud Canvas (p. 2116).

**PlayerAccountSample Tutorial**

To test the Player Account Cloud Gem

1. In the Project Configurator, select the CloudGemSamples project.
2. Start Lumberyard Editor.
3. On the Welcome to Lumberyard Editor screen, click Open level, Levels, PlayerAccountSample, Open.
4. Click AWS, Cloud Canvas, Select a Deployment.
5. Click the deployment that you want to use, and then click OK.
6. Press Ctrl+G to start the game.
7. In the main menu of the sample, click Create Account.

![Player Account Menu](image)

**Note**

If the Create Account option does not appear, click Sign Out and try again.

8. Enter a user name, password, and an email account to which you have access.

```
Username: TestUser
Password: ********
Email: 
```

9. Click Create.
10. From the email account, copy the confirmation code that you receive.
11. In Lumberyard, paste the confirmation code into the Verification Code box, and then click Verify.

```
Username: TestUser
Verification Code: 123456
```

Version 1.28

2194
12. Sign in using the password that you specified earlier. This tests the custom authentication flow.

13. Click **Manage Account**.

The main menu displays the player name that you specified.

14. Click **Edit Account**.

15. Type in a name for **Player Name**, and then click **Save**. This step tests the player service API.
16. Press ESC to stop the game.

17. Open a command prompt to your Lumberyard dev directory.

18. Use the `show-players` command to verify the player has been created.

```bash
$ lmbr_aws player-account show-players
```

```json
{
    "AccountId": "b8103676-1234-5678-9abc-def012345678",
    "CognitoUsername": "TestUser",
    "IdentityProviders": {
        "Cognito": {
            "IdentityProviderId": "Cognito",
            "create_date": 1613433006.709,
            "email": "mary@example.com",
            "enabled": True,
            "gender": "None",
            "last_modified_date": 1613433006.709,
            "status": "FORCE_CHANGE_PASSWORD",
            "username": "Mary Major"
        }
    }
}
```

19. Use the `edit-player` command to change the player name, to test the administrative service API. Replace the account ID shown in the example below with the actual account ID from the output of the previous step.

```bash
$ lmbr_aws player-account edit-player --account-id b8103676-1234-5678-9abc-def012345678 --playername "Major Mary"
```

20. Use the `show-player-details` command to verify the change in the profile for the player. Replace the account ID shown in the example below with the actual account ID of the player used in the previous step.

```bash
$ lmbr_aws player-account show-player-details --account-id b8103676-1234-5678-9abc-def012345678
```

Integrating the Player Account Cloud Gem into an Existing Project

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To integrate the Player Account Cloud Gem into an existing project, you must enable the Player Account Cloud Gem, update your deployment stack, and integrate the gem into your game menu.
To integrate the Player Account Cloud Gem into an existing project

1. Enable the gem. In the Project Configurator, click **Cloud Gem Player Account**.

2. **Create or update the project stack (p. 2350).**

   The Player Account Cloud Gem uses an Amazon Cognito user pool.

3. **Create or update the deployment stack (p. 2352).** Be sure the CloudGemPlayerAccount resource group is added to your deployment. If your deployment doesn't have the CloudGemPlayerAccount resource group, make sure that the Cloud Gem Player Account gem is enabled in the Project Configurator.

4. At a command prompt, enter the following command to update the deployment access stack.

   ```
   lmbr_aws deployment update-access
   ```

   This ensures that the CloudGemPlayerAccount resource group is correctly linked to the PlayerAccess identity pool in the deployment access stack.

5. Integrate the gem into your game menu. We recommend that you include the following features:

   - **A global signout** – This security feature allows players to invalidate an account's tokens across all devices if a device has been lost or stolen. The sample level shows how to offer this as a standalone feature. If the previous password was compromised and already used to obtain authentication tokens, you can also have the menu automatically globally sign out after a password change. To implement this functionality, do the following:
     1. Monitor the CloudGemPlayerAccountNotifications::OnChangePasswordComplete EBus event.
     2. When you receive confirmation that the password change has succeeded, send the following event.

       ```
       EBUS_EVENT(CloudGemPlayerAccountRequestBus, GlobalSignOut, username)
       ```

   - **Email verification** – An email verification step ensures that email can be used to recover account access if a password has been forgotten. You can customize the content of the verification emails by using Lambda triggers. To add a new Lambda trigger to the AWS CloudFormation template, use the LambdaConfig property of PlayerUserPool in the dev\Gems\CloudGemPlayerAccount\AWS\resource-template.json file. For more information, see Customizing User Pool Workflows by Using AWS Lambda Triggers.

Building the Menus for the Player Account Cloud Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To build the menus, use the code for CloudGemPlayerAccountRequestBus and CloudGemPlayerAccountNotificationBusHandler in the dev\Gems\CloudGemPlayerAccount\Code\Include\CloudGemPlayerAccount\CloudGemPlayerAccountBus.h file. Keep in mind the following points:

- Most of the EBus events are wrappers for Amazon Cognito user pool API operations. For the Amazon Cognito API reference, see Amazon Cognito Identity Provider.
• To return the user name for the currently logged-in player, call the `GetCurrentUser` function, which has a corresponding `OnGetCurrentUserComplete` event. This function and EBus event are not part of the Amazon Cognito user pool API but are needed for most calls to the API.

• The `GetCurrentUser` function also loads the credentials that were returned and caches them inside the gem's system component. As a result, subsequent calls to `GetCurrentUser` are faster.

• The act of signing in also caches the player's credentials inside the gem's system component.

For information about Player Account Cloud Gem functions for creating accounts, password recovery, signing in and out, and account management, see Player Account Cloud Gem Implementation Details (p. 2199).

Storing Data Associated with a Player Account

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Each player account has a randomly generated account ID. When you store data associated with a player, use this ID instead of the identifiers provided by the Amazon Cognito user and identity pools. You should do this for the following reasons:

• Lumberyard uses the `PlayerAccounts` table defined in the `dev\Gems\CloudGemPlayerAccount\AWS\resource-template.json` file to map account IDs to Amazon Cognito IDs. By associating a player's data with the account ID, you can move all of the player's data from one identity to another by simply updating the mapping.

• The Amazon Cognito ID on the Amazon Cognito identity pool is not guaranteed to stay the same for a particular player. For example, it can change if identities are merged. Amazon Cognito identity pools are public facing and do not prevent the merging of identities.

• If you configure the `PlayerAccess` identity pool to support identity providers other than the Amazon Cognito user pool, the user pool's user name cannot be used for players that use another provider.

• If a user name is deleted from the Amazon Cognito user pool, the name can be reused. Amazon Cognito user pools are publicly visible and do not prevent a player from deleting his or her own user name.

A `PlayerName` field is provided in the `PlayerAccounts` table as an example of how to store player data for an account. The Amazon Cognito user pool supports storing a variety of attributes and could also be used for storing player data. However, this is not recommended if you configure the `PlayerAccess` identity pool to use more than one identity provider. In such a scenario, some players might not have a user in the Amazon Cognito user pool.

**Warning**
Do not store the data for unauthenticated (guest or anonymous) players who are using an Amazon Cognito ID directly or indirectly on the server. The reasons not to store this data are the following:
• A person who knows the Amazon Cognito ID for an unauthenticated account can claim the identity by linking it to a user that the person controls. The identity's original owner cannot recover the account if an original device is lost or the identity is stolen. For more information on identity pools, see Identity Pools.

• The local copy of the unauthenticated Amazon Cognito ID is overwritten with a new one when the player logs in and out.

• The player has no way to recover an unauthenticated Amazon Cognito ID after it has been overwritten locally. The identity still exists in the identity pool, but the identity and data associated with its Amazon Cognito ID effectively become orphaned.

• The buildup of orphaned identities can be misleading when trying to determine the actual number of players.

• If a player uses an unauthenticated identity and then creates a new account, the account ID from the unauthenticated identity does not carry over to the new account.

**Obtaining Player Account IDs**

You can use the following techniques to obtain player account IDs and account information programmatically.

**To get a player's account ID in game** – Call GetPlayerAccount on the CloudGemPlayerAccountRequestBus. This EBus is defined in the dev\Gems\CloudGemPlayerAccount\Code\Include\CloudGemPlayerAccount\CloudGemPlayerAccountBus.h file. The player's account ID is in the AccountResultInfo parameter of CloudGemPlayerAccountNotificationBusHandler::OnGetPlayerAccountComplete.

**To get the account ID from a Lambda function located behind an API Gateway** – Because Amazon API Gateway provides the Amazon Cognito ID to the Lambda function, the Lambda function can look up the account ID. To quickly look up the account ID, you can use the global secondary index CognitoIdentityIdIndex of the PlayerAccounts table. The PlayerAccounts table is defined in the dev\Gems\CloudGemPlayerAccount\AWS\resource-template.json file.

**To get the caller's account information without relying on the caller to provide its account ID** – Use the request's authentication parameters. For an example, see the code in the dev\Gems \CloudGemPlayerAccount\AWS\lambda-function-code\api\account.py file.

**Player Account Cloud Gem Implementation Details**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the API operations described on this page to manage player accounts with the Player Account Cloud Gem (p. 2188).
Functions and Callbacks

Most functions for the Player Account Cloud Gem have a corresponding callback that supplies the results of the call. The results contain the information gathered or error information. To make calls to and receive responses from player registration and login component API operations, you use the EBus (p. 1851).

The callbacks use this naming convention: For every function X, the corresponding callback function is named OnXComplete. For example, when a response to an EBus call of SignUp() is ready, the response is sent through the EBus to OnSignUpComplete(). For more examples, see dev\Gems\CloudGemPlayerAccount\Code\Include\CloudGemPlayerAccount \CloudGemPlayerAccountBus.h.

HasCachedCredentials() is an exception to this convention because it is a synchronous call and has no corresponding handler.

Functions for Key Player Account Cloud Gem Tasks

You can use the functions described in this section for new account creation, password recovery, player sign-in, and account management. Most of the source code for the classes and functions can be found in the Lumberyard directory dev\Gems\CloudGemPlayerAccount\Code\Source.

Creating an Account (CloudGemPlayerAccountBus)

Use the following functions for creating new accounts.

SignUp(username, password, attributes)

Creates an account. The attributes parameter contains relevant data (for example, email, address, name, and profile). A confirmation email or text message is sent to the player as part of the signup. Password recovery requires a valid email address.

OnSignUpComplete(resultInfo, deliveryDetails, wasConfirmed)

A callback function that returns the results (p. 2201) of the signup. The deliveryDetails (p. 2202) attribute contains details about where the confirmation code was sent.

ResendConfirmationCode(username)

Resends the confirmation code to the player. The confirmation code received from the player is passed as an argument to ConfirmSignUp(username, confirmationCode).

Password Recovery (CloudGemPlayerAccountSystemComponent)

Use the following functions if a player has forgotten a password and has an account with a valid email or phone number.

ForgotPassword(username)

Causes a confirmation code to be sent to the player.

OnForgotPasswordComplete(resultInfo, deliveryDetails)

Contains information regarding the password reset.

ConfirmForgotPassword(username, password, confirmationCode)

Receives the confirmation code, user name, and new password from the player and sets the new password.

Sign In and Sign Out

Use the following functions to sign players in and out of your game.
InitiateAuth(username, password)

Call this function to sign in players. Sign-in is a two-step process. When InitiateAuth finishes, the server issues a challenge to which the client must respond correctly. Currently, authorization by username and password is supported; MFA and other forms of additional authentication are not.

You have two ways to sign out: SignOut(username) and GlobalSignOut(username).

SignOut

Clears all cached authentication information from memory, so that the player's credentials are essentially forgotten. No server-side changes occur, so the authentication information, if it were known, could still be used. The SignOut function always succeeds, even if the player is not actually signed in.

GlobalSignOut

Invalidates all access and refresh tokens for the player on all devices that the player might be signed into. Unlike SignOut, GlobalSignOut can succeed only when the player is signed in.

Account Management (CloudGemPlayerAccountSystemComponent)

All of the following functions require that the player be signed in.

ChangePassword(username, previousPassword, proposedPassword)

Allows a player to update a password without the use of a confirmation code.

DeleteOwnAccount(username)

Deletes the player's account. Use this function with caution.

GetUser(username)

Retrieves attributes (for example, email, address, name, and profile) that have been associated with the player's account.

UpdateUserAttributes(username, attributes)

Updates the specified set of attributes. If any updates require verification, confirmation codes are sent. Details are found in the deliveryDetailsArray parameter that is passed to OnUpdateUserAttributesComplete. Note that it is possible to update attributes that previously did not have any associated information.

VerifyUserAttribute(username, attributeName, confirmationCode)

Following a call to UpdateUserAttributes, verifies a single attribute that requires confirmation. Most attribute updates do not require verification.

DeleteUserAttributes(username, attributesToDelete)

Removes the specified set of attributes from the player's account. The attributes can be added again with a call to UpdateUserAttributes.

Key Player Account Cloud Gem Classes and Functions

This section describes key classes and functions in the Player Account Cloud Gem.

BasicResultInfo Class

The BasicResultInfo class bundles together information returned in almost all responses from the component. The class has no functions to be used at run time and contains the following values, which are public:
wasSuccessful

A Boolean that indicates whether the request succeeded.

username

The name of the player for which a request was made.

errorTypeName

The name of the error type.

errorTypeValue

The numeric value of the error type. If errorTypeValue is greater than or equal to 0, it corresponds to a standard CognitoIdentityProviderErrors value. Otherwise, an unexpected error occurred. Most commonly, the player has not yet signed in for the operation. In this case, see the error information in errorTypeName for details.

errorMessage

A human readable string that describes the error.

DeliveryDetails Class

The DeliveryDetails class provides details related to the sending of confirmation codes to the player. It contains the following string functions:

GetAttributeName()

The name of the attribute for which a confirmation code was sent (for example, email).

GetDeliveryMedium()

The medium used to send the confirmation code (for example, EMAIL).

GetDestination()

The destination to which the confirmation code was sent. The destination is partially obscured for security (for example, d***@a***.com).

DeliveryDetailsArray Class

The DeliveryDetailsArray class is a collection of DeliveryDetails objects and contains the following functions:

GetSize()

The number of DeliveryDetails objects in the array.

At(index)

Returns a copy of the object at the specified index.

UserAttributeValues Class

UserAttributeValues maps attribute names like address, email, and family_name to attribute values. An attribute value can be a string up to 256 characters. In general, the attributes are those used by the OpenID Connect specification. See the UserAttributeValues.h file for details.

Phone numbers must follow these formatting rules:
• The phone number must start with a plus (+) sign, followed immediately by the country code.
• The phone number can contain only the + sign and digits.
• You must remove characters such as parentheses, spaces, or dashes from the phone number before submitting the value. For example, a United States–based phone number must follow the format +14325551212.

Resource Group

The Player Account Cloud Gem uses an Amazon Cognito user pool that contains all registered players. It is set as an authentication provider with PlayerAccessIdentityPool. For more information, see Controlling Access to Resources (p. 2370).

Speech Recognition Cloud Gem

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Speech Recognition Cloud Gem to add speech recognition and natural language processing to your Lumberyard game. The Speech Recognition Cloud Gem uses the Amazon Lex service, which recognizes the intent of spoken player input so that your game can react accordingly. Your players can use natural language and do not have to memorize or use specific phrases to initiate commands.

Text to Speech Cloud Gem (Using Amazon Polly)

This cloud Gem is deprecated and no longer supported. Some functionality is broken because of its dependency on the Cloud Gem Portal, which was removed in Lumberyard 1.28. Complete documentation for this Gem from previous versions of Lumberyard can be found in the Documentation Archive.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Text-to-Speech (TTS) Cloud Gem to enhance your gameplay and workflows with synthesized speech. The Cloud Canvas Text-to-Speech (TTS) Cloud Gem uses Amazon Polly, which is a text-to-speech service that turns text into lifelike speech. Amazon Polly offers dozens of lifelike voices in a variety of languages. The service also creates lip synchronization from the text that you provide. You can import the generated audio and speech mark files into your dialogue system. Currently, the Text-to-Speech Gem supports playback of PCM (pulse-code modulation) files.

You can use the text-to-speech service in two ways:

• You can prepackage speech content and include it with your game so that your clients can access it immediately.
- Your clients can invoke the Amazon Polly service to provide text to speech while your game is running.

In the first approach, you prepare voices and dialogues that players require and store them on the client. This removes the need for the client to connect to the backend to generate and download lines that are known to be necessary. The trade-off is that the client must store the files locally.

**Web Communicator Cloud Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Web Communicator cloud gem to inform your game's connected clients, editors, or servers of events from other cloud gems. Your game clients, editors, or servers can use this information to take action (such as updating themselves) without restarting or polling AWS services for updates.

The following diagram shows how the Web Communicator cloud gem connects cloud gems like Dynamic Content to the receivers of the events.

Cloud gems like Dynamic Content send events through Lumberyard's cross-gem communication system (p. 2319) to the Web Communicator cloud gem. The Web Communicator cloud gem sends the event to the client through AWS IoT, which maintains a persistent socket connection with the client. Your game can use this channel of communication to notify the client that new content is available. The client can then download the new content or update itself as needed.
Note
You can publish and receive from your game client, editor, or server. However, to reduce potential expenses, the default implementation on all channels is receive only.

Publishing Messages

You can publish messages directly through AWS IoT or send them through the Web Communicator API first for validation. Each method is useful for the following applications:

- **Directly through AWS IoT** – For communication between trusted parties (for example, from editor to editor).
- **Web Communicator API** – For game client messages that require validation before publishing.

You can use the AWS IoT console or the game client to subscribe and listen to messages. For more information on using this console to view messages, read View MQTT messages with the AWS IoT MQTT client. You can use the `lmbr_aws web-communicator list-channels` and `list-users` commands to find the subscription topic to listen to.

User ID and Status Table

The Web Communicator cloud gem supports a simple DynamoDB table that records the Amazon Cognito Identity and status of registered users. The `lmbr_aws` CLI provides commands to list users and modify their status. Setting a user’s status to “banned” revokes the user’s AWS IoT policy and prevents the user from connecting or receiving further messages.

Connection Types

The Web Communicator cloud gem can use WebSocket or OpenSSL connections.

WebSocket Connections

Web Communicator uses WebSocket connections and Amazon Cognito identities by default. WebSocket connections, which use Message Queuing Telemetry Transport (MQTT) clients and AWS Signature Version 4, attach the AWS IoT policy to the Amazon Cognito Identity of the user. AWS Signature Version 4 is a protocol for authenticating inbound API requests to AWS services over WebSocket connections. For more information, see Authenticating Requests (AWS Signature Version 4) in the AWS General Reference.

For samples of Amazon Cognito and AWS IoT policies that Web Communicator creates, see Web Communicator Cloud Gem Authentication and AWS Policies.

OpenSSL Connections

OpenSSL connections, which are commonly used for embedded devices, use the MQTT protocol and certificate authentication. In OpenSSL connections, each device has a certificate and a private key that are used for encrypted communications.

You can use the `lmbr_aws web-communicator register-client` command to generate the certificates and keys that OpenSSL requires. The client can connect through OpenSSL after the certificates and keys are installed in the `@user@certs/aws` directory. The Web Communicator service attaches the appropriate policy to the certificate. The certificate and private key are returned to the client and cached on the client.

Note
By default, only WebSocket connections can be registered directly from the client. This behavior can be changed in the `client_request.py` request_registration call.
AWS IoT Pricing

AWS IoT prices connectivity per million minutes of connection or per device per year. Message pricing is per million messages and decreases with volume pricing. For more information, see AWS IoT Core Pricing.

Topics

- Using the Web Communicator CLI (p. 2206)
- Testing the Web Communicator Cloud Gem (p. 2207)
- Communicating to Clients from a Cloud Gem (p. 2213)
- Web Communicator Cloud Gem Authentication and AWS Policies (p. 2218)
- Web Communicator Cloud Gem Service API (p. 2219)

Using the Web Communicator CLI

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you enable the Web Communicator Cloud Gem, associated commands become available in the `lmbr_aws` CLI. You can then use these commands to register and manage client connections, and send messages to those clients.

The following `lmbr_aws` CLI extension commands are enabled by the Web Communicator Cloud Gem.

**web-communicator list-channels**

List all existing channels.

The Common Arguments (p. 2391) can also be used with this command.

**web-communicator list-users**

List all existing users.

The Common Arguments (p. 2391) can also be used with this command.

**web-communicator register-client**

Register an OpenSSL or WebSocket client connection. Saves the device cert, port, endpoint, and private keys to the following files in the current or specified folder:

- `webcommunicatorkey.pem`
- `webcommunicatordevice.pem`
- `deviceInfo.json`

These files should be placed in the user storage directory for your application or platform. For example, on a PC, the directory is `lumberyard_version\dev\Cache\project_name\pc\user\certs\aws`.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

- `--type <OPENSSL | WEBSOCKET>`
  
  Type of connection to register.
• --path <folder_path>
  (Optional) Output folder in which to save the registration files. Defaults to the current directory.

**web-communicator send-message**

Send a message to all clients on a channel, or to a specific user.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --channel-name <channel>
  Channel name on which to send the message.
• --message <message>
  Message to broadcast. Maximum 1024 characters.
• --client-id <client_ID>
  (Optional) Amazon Cognito account ID of the client to whom the message should be sent. If this is omitted, the message will be broadcast to all clients on the channel.

**web-communicator set-user-status**

Set the user status of a client to either "banned" or "registered".

• --client-id <client_ID>
  Amazon Cognito account ID of the client for which the status should be set.
• --status <REGISTERED | BANNED>
  Status to set.

**web-communicator show-logs**

Show recent log events.

In addition to the Common Arguments (p. 2391), this command accepts the following arguments:

• --minutes <minutes>
  (Optional) Display log events for the specified number of minutes before now. The default is 10 minutes.

**Testing the Web Communicator Cloud Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To perform a simple test of the Web Communicator cloud gem, you can use the CommunicatorSample level that is included with Lumberyard.

**Prerequisites**

This tutorial assumes the following:
Lumberyard User Guide Lumberyard User Guide
Cloud Gems

- Your Lumberyard project has the Web Communicator and Player Account cloud gems enabled (in the Project Configurator, choose Cloud Gem Web Communicator and Cloud Gem Player Account).
- You used the Cloud Canvas Resource Manager or the lmbr_aws command line tool (p. 2390) to create the following:
  - A project stack (p. 2350) for your project.
  - A deployment (p. 2352) for your project that includes the CloudGemWebCommunicator and CloudGemPlayerAccount resource groups.
- You have an email account for testing.

Test Workflow Summary

The test workflow uses the PlayerAccountSample level, the CommunicatorSample level, and the AWS IoT management console in the following ways:

A. PlayerAccountSample – To create an player account (authenticated identity) for testing. Web Communicator uses authenticated identities by default.
B. CommunicatorSample – To register a WebSocket, connect the WebSocket, and request a channel list.
C. AWS IoT management console – To publish a test message that appears in the CommunicatorSample level.

Test Workflow Steps

The following sections describe each of the workflow steps in detail.

A. Using PlayerAccountSample to Create a Player Account

In this step, you use the PlayerAccountSample level to create a player account and sign in with it.

To create a player account for the test

1. Follow Testing the Player Account Cloud Gem (steps 1-11) to create a player account. To verify the account, submit the verification code that is emailed to you.
2. With the PlayerAccountSample level still running, sign in to the PlayerAccountSample level with the credentials that you created (Testing the Player Account Cloud Gem step 12).
3. Press ESC to exit the level.

B. Using CommunicatorSample to Register and Connect a Websocket

In this step, register and connect a Websocket that can receive messages from AWS IoT.

To register and connect a Websocket

1. In Lumberyard Editor, choose File, Open.
2. In the Open a Level dialog box, choose CommunicatorSample, and then click Open. If you are using a launcher and need to restart, your authenticated identity should be found on startup.
3. Press Ctrl+G to start the level. The sample level looks similar to the following image.
4. Click **Register WebSocket**.

The status changes to **Registered**.

5. Click **Connect WebSocket**.
6. Click **Request List**.

The status changes to **Connected**.
Your test channels are listed in the sample level user interface.

7. Leave the CommunicatorSample level running.

Now that you have registered and connected a WebSocket, you are ready to test it.

C. Using the AWS IoT Management Console to Publish a Test Message

In this step, you send a test from the message AWS IoT console to the CommunicatorSample level.

To send a test message to the CommunicatorSample level

1. Do one of the following:
   * Press Alt+Tab to change to your browser.
   * Open a browser on another computer that has internet access.

2. In your browser, sign in to the AWS Management Console with the AWS credentials that you used to create your project stack and deployment.

3. Choose Services, Internet of Things, IoT Core to open the AWS IoT console.

4. In the left navigation pane of AWS IoT console, choose Test.
5. Navigate to the **Publish** section at the bottom of the page.

Publish

Specify a topic and a message to publish with a QoS of 0.

cgs624803/dep/CloudGemWebCommunicator

```
{
  "Channel": "CloudGemWebCommunicator",
  "Message": "This is a test"
}
```
6. In the **Publish** section, enter the following information:

   a. For **Specify a topic**, specify your channel as `project_name/deployment_name/CloudGemWebCommunicator` (for example, `cgs624803/dep/CloudGemWebCommunicator`).

   b. For **Message**, add a message that uses the following syntax:

   ```json
   {      
   "Channel":"CloudGemWebCommunicator",      
   "Message":"This is a test"      
   }
   ```

7. Choose **Publish to topic**.

8. Return to the CommunicatorSample level to see the results. If you are on the same computer, press **Alt+Tab** to return to Lumberyard Editor.

   The sample level shows the message that you sent from AWS IoT.

   ![Communicating to Clients from a Cloud Gem](image)

**Communicating to Clients from a Cloud Gem**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To send messages to the Web Communicator service, your cloud gem can use the default channel that the Web Communicator cloud gem provides or a channel that you define. Web Communicator can validate the message type and pass the message to AWS IoT for distribution to one or more clients.

**Note**

Because the caller of a cross-gem request cannot be identified, any cloud gem can make a request to broadcast on any valid channel. For this reason, you should consider the behavior of any gems in your project that implement cross-gem communication.

To have your cloud gem send events through Web Communicator to connected clients, perform the following steps:

1. (Optional) Define Broadcast and/or Private Channels (p. 2214)
2. Add the Web Communicator Service Interface ID to Your Gem’s resource-template.json File (p. 2214)

3. Send the Lambda Request from Your Service (p. 2215)

4. Handle the Response on the Client (p. 2216)

1. (Optional) Define Broadcast and/or Private Channels

You can optionally define your own channels in your cloud gem’s resource-group-settings.json file. A channel consists of a unique name, a type (BROADCAST or PRIVATE) and an optional CommunicationChannel. To reduce the number of subscriptions required on the server, the communication channel system allows your channel to make use of a base channel by embedding your actual channel name in the message. When messages are broadcast from other cloud gems as EBus (p. 1851) events, the channel name is embedded in the message and parsed on the client.

To declare broadcast and/or private channels, add them to your cloud gem's resource-group-settings.json file in the GemSettings, CloudGemWebCommunicator, Channels section. The following example is from the Dynamic Content (p. 2155) cloud gem.

```json
{
   "GemSettings":
   {
      "CloudGemWebCommunicator":
      {
         "Channels":
         [
            {
               "Name": "CloudGemDynamicContent",
               "Types": ["BROADCAST","PRIVATE"],
               "CommunicationChannel": "CloudGemWebCommunicator"
            }
         ]
      }
   }
}
```

The relevant fields are as follows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specifies the name of the channel.</td>
</tr>
<tr>
<td>Types</td>
<td>The type of channel that you want to enable. This can be BROADCAST, PRIVATE, or both.</td>
</tr>
<tr>
<td>CommunicationChannel</td>
<td>The name of the base WebCommunicator channel (for example, CloudGemWebCommunicator). For purposes of efficiency, the channel that actually carries the message is packed into the base channel. The name of the carried channel (in this example, CloudGemDynamicContent) is embedded in the message itself. Lumberyard unpacks this name transparently on the client side for the EBus handler.</td>
</tr>
</tbody>
</table>

2. Add the Web Communicator Service Interface ID to Your Gem’s resource-template.json File

The request API between your cloud gem and the Web Communicator gem is built on the cross-gem communication system (p. 2319). The interface for the Web Communicator's broadcast and send API operations are defined in the lumberyard_version\dev\Gems\CloudGemWebCommunicator\AWS\api-definition\sendmessage_1_0_0.json file.
To make this interface callable from your cloud gem's service Lambda function, you add an InterfaceID property to the lumberyard_version\dev\Gems\gem_name\AWS\resource-template.json file.

In the ServiceLambdaConfiguration section, under Properties, add a Services section entry like the following.

```
"Services": [
  {
    "InterfaceId": "CloudGemWebCommunicator_sendmessage_1_0_0",
    "Optional": "True"
  }
]
```

**Note**

If you expect the Web Communicator to be continuously available, set the Optional property to False.

For more information, see the Using an Interface (p. 2322) section of the cross-gem communication topic.

### 3. Send the Lambda Request from Your Service

To send a message from your cloud gem to one or more clients, your cloud gem's Lambda code makes a call to the Web Communicator. The call includes the message that you want to either broadcast to multiple clients or send to an individual user.

First, in the Python module from which you want to make the messaging request, specify the following import statements.

```
import cgf_lambda_settings
import cgf_service_client
```

#### Broadcasting a Message

To broadcast a message, use the `__send_communicator_broadcast` function and pass the message text in the message parameter, as in the following example.

```python
def __send_communicator_broadcast(message):
    interface_url =
    cgf_lambda_settings.get_service_url("CloudGemWebCommunicator_sendmessage_1_0_0")
    if not interface_url:
        print('Messaging interface not found')
        return

    client = cgf_service_client.for_url(interface_url, verbose=True, session=boto3._get_default_session())
    result = client.navigate('broadcast').POST({"channel": "CloudGemDynamicContent", "message": message})
```

#### Sending a Message Directly to a Client

To send a message directly to a client, use the `__send_communicator_direct` function. As before, pass the message text in the message parameter, but this time specify the ID of a specific client in the client_id parameter.

```python
def __send_communicator_direct(message, client_id):
    interface_url =
    cgf_lambda_settings.get_service_url("CloudGemWebCommunicator_sendmessage_1_0_0")
    if not interface_url:
print('Messaging interface not found')
return

client = cgf_service_client.for_url(interface_url, verbose=True,
session=boto3._get_default_session())
result = client.navigate('send', client_id).POST({"channel": "CloudGemDynamicContent",
"message": message})

4. Handle the Response on the Client

To handle the response on the client, subscribe manually or automatically to messages from Web Communicator, and then create a handler that processes the incoming messages.

Requesting a Connection

To request a connection, use the RequestConnection function.

CloudGemWebCommunicatorRequestBus.Broadcast.RequestConnection("connection_type")

For connection_type, specify either WEBSOCKET or OPENSSL.

Requesting a Registration

To request a registration, use the RequestRegistration function.

CloudGemWebCommunicatorRequestBus.Broadcast.RequestRegistration("connection_type")

For connection_type, specify either WEBSOCKET or OPENSSL.

Subscribing to Channels

You can use the RequestChannelList function to request and subscribe to the channels to which the client listens.

Subscribing to All Channels

To request and subscribe to all the channels that the client requires, use the RequestChannelList function.

CloudGemWebCommunicatorRequestBus.Broadcast.RequestChannelList()

You can also subscribe individually to channels, but using the RequestChannelList function is more straightforward and should be considered the standard approach.

Subscribing to Individual Channels

To subscribe to channels manually, use the RequestSubscribeChannel function and specify the name of the channel in the channelName parameter, as in the following example.

CloudGemWebCommunicatorRequestBus.Broadcast.RequestSubscribeChannel(channelName)

Note

If you specified the CommunicationChannel property in the channel definitions step (p. 2214) (that is, you are using a base channel), specify that value for the channelName parameter.

Unsubscribing from a Channel

To unsubscribe from a channel, use the RequestUnsubscribeChannel function and specify the name of the channel in the channelName parameter, as in the following example.
CloudGemWebCommunicatorRequestBus.Broadcast.RequestUnsubscribeChannel(channelName)

**Disconnecting**

To disconnect, use the `RequestDisconnect` function.

CloudGemWebCommunicatorRequestBus.Broadcast.RequestDisconnect()

**Create a Message Handler**

After you subscribe the client to one or more channels, create a message handler that processes the messages from Web Communicator.

**In Lua**

In Lua, in an `OnActivate` call, set up an EBus connection to listen for Web Communicator updates:

```lua
if CloudGemWebCommunicatorUpdateBus ~= nil then
    Debug.Log("Listening for communicator updates")
    self.communicatorUpdateBus = CloudGemWebCommunicatorUpdateBus.Connect(self,
        WebCommunicator.ModuleEntity)
else
    Debug.Log("Web Communicator not found")
end
```

Next, declare your message handler, as in the following example.

```lua
function DynamicContentTest:MessageReceived(channelName, messageData)
    if self:IsDynamicContentUpdate(channelName) then
        Debug.Log("DynamicContent update received: " .. messageData)
        DynamicContentRequestBus.Event.HandleWebCommunicatorUpdate(DynamicContent.ModuleEntity,
            messageData)
    end
end
```

For a sample Lua script, see `lumberyard_version\dev\CloudGemSamples\Scripts \CommunicatorSample.lua`.

**In C++**

In C++, declare a component handler class like the following:

```cpp
class SomeHandler :
public CloudGemWebCommunicator::CloudGemWebCommunicatorUpdateBus::Handler
virtual void ConnectionStatusChanged(const AZStd::string& connection) = 0;
virtual void MessageReceived(const AZStd::string& channelName, const AZStd::string& channelMessage) = 0;
virtual void RegistrationStatusChanged(const AZStd::string& registrationStatus) = 0;
virtual void SubscriptionStatusChanged(const AZStd::string& channelName, const AZStd::string& subscriptionStatus) = 0;
Connect:
void SomeHandler::Activate()
{
    CloudGemWebCommunicator::CloudGemWebCommunicatorUpdateBus::Handler::BusConnect();
}
Handle:
void SomeHandler::MessageReceived(const AZStd::string& channelName, const AZStd::string& channelMessage)
```
Resources

For more information, see the following:

- For a sample level, see the CommunicatorSample (p. 2207) sample level in the CloudGemSamples project.
- For more information about cross-gem communication in Cloud Canvas, see Cross-Gem Communication (p. 2319).
- For the Web Communicator cloud gem API, see Web Communicator Cloud Gem Service API (p. 2219).
- For the Web Communicator cloud gem source code, see the files in the lumberyard_version\dev\Gems\CloudGemWebCommunicator\Code\ directory and its subdirectories.

Web Communicator Cloud Gem Authentication and AWS Policies

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The default Web Communicator authentication system uses authenticated Amazon Cognito users with AWS Signature Version 4. For more information, see Authenticating Requests (AWS Signature Version 4) in the AWS General Reference.

Note
Lumberyard advises against implementing unauthenticated access. If you surrender control over which users can connect to your AWS IoT network, you can incur unexpected costs.

Connection to AWS IoT requires two policies, which Web Communicator generates for you:

- A PlayerAccess role that can connect to, subscribe to, and receive messages from any AWS resource that is added to Web Communicator.
- An AWS IoT policy that allows authenticated users to use their Amazon Cognito Identity as their client ID for connection purposes. The policy grants the aggregate channel the permissions that have been requested from the other cloud gems.

You do not have to make changes to these generated policies to use them. The following sections provide reference samples of the generated policies if you want to modify them.

Amazon Cognito Policy Sample

The following sample Amazon Cognito policy grants the IotPlayerPermissions1 user permissions to connect, subscribe, and receive.

```json
{
    "Action": [
        "iot:Connect",
        "iot:Subscribe",
        "iot:Receive"
    ],
    "Resource": [
        "*
    ],
    "Effect": "Allow",
```

Version 1.28
2218
AWS IoT Policy Sample

The following sample AWS IoT policy is attached to a client. The final AWS IoT policy is an aggregation of channels from the cloud gems that the client uses.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["iot:Connect"],
      "Resource": [
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["iot:Receive"],
      "Resource": ["*" ]
    },
    {
      "Effect": "Allow",
      "Action": ["iot:Subscribe"],
      ]
    }
  ]
}
```

Web Communicator Cloud Gem Service API

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Web Communicator cloud gem service API contains the following general requests and client requests. These calls are exposed on the client by the code in `lumberyard_version\dev\Gems\CloudGemWebCommunicator\Code\AWS\ServiceApi\CloudGemWebCommunicatorClientComponent.*`. You do not have to manage these calls directly on the client.

- GET `/service/status` (p. 2220)
- GET `/client/channels` (p. 2220)
General Requests

The Web Communicator cloud gem service API contains a request to get service status.

GET /service/status

Returns the service's status. This request is useful for testing connectivity.

Client Requests

GET /client/channels

Requests a list of channels available to the client. Returns a ChannelRequestResult object.

The ChannelRequestResult object contains a ChannelRequestResultArray object

The ChannelRequestResultArray object contains a ChannelInfo object.

### ChannelInfo Object

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelName</td>
<td>String</td>
<td>The name of the channel to which to connect.</td>
</tr>
<tr>
<td>CommunicationChannel</td>
<td>String</td>
<td>The name of the communication channel.</td>
</tr>
<tr>
<td>CommunicationType</td>
<td>String</td>
<td>Can be one of RECEIVE, SEND, or BOTH.</td>
</tr>
<tr>
<td>Subscription</td>
<td>String</td>
<td>The name of the subscription.</td>
</tr>
</tbody>
</table>

POST /client/channel

Posts a message to a client channel.

The request parameter is a BroadcastRequest object.

### BroadcastRequest Object

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelName</td>
<td>String</td>
<td>The channel to which to publish.</td>
</tr>
<tr>
<td>Message</td>
<td>String</td>
<td>The message to send.</td>
</tr>
</tbody>
</table>

GET /client/registration/\{registration_type\}

Requests the registration of a new client. Returns a RegistrationResult object and a ChannelRequestResult object.

Request Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registration_type</td>
<td>String</td>
<td>Type of connection that the client uses. Specify OPENSSL or WEBSOCKET.</td>
</tr>
</tbody>
</table>
### RegistrationResult Object

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionType</td>
<td>String</td>
<td>Can be OPENSSL or WEBSOCKET.</td>
</tr>
<tr>
<td>DeviceCert</td>
<td>String</td>
<td>For OpenSSL connections, a one time only device certificate.</td>
</tr>
<tr>
<td>Endpoint</td>
<td>String</td>
<td>The AWS IoT connection address.</td>
</tr>
<tr>
<td>EndpointPort</td>
<td>Double</td>
<td>The AWS IoT connection port.</td>
</tr>
<tr>
<td>PrivateKey</td>
<td>String</td>
<td>For OpenSSL connections, a one time only private key.</td>
</tr>
<tr>
<td>Result</td>
<td>String</td>
<td>Can be DENIED or SUCCESS.</td>
</tr>
</tbody>
</table>

The `ChannelRequestResult` object contains a `ChannelRequestResultArray` object.

The `ChannelRequestResultArray` object contains a `ChannelInfo` object.

For the contents of the `ChannelInfo` object, see GET /client/channels (p. 2220).

### Cloud Canvas Gameplay Design and Engineering Guide

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Cloud Canvas's deployments, resource groups, and cloud gems (provided by yourself or others) to design and engineer the cloud connected features of your game.

**Topics**

- Cloud Canvas Tools in Lumberyard Editor (p. 2221)
- Managing Cloud Canvas Profiles (p. 2223)
- Using Resource Manager in Game Design (p. 2224)

### Cloud Canvas Tools in Lumberyard Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The tools in Lumberyard Editor make it easy for you to connect your game to AWS. To get started, click AWS in the Lumberyard Editor menu bar:
The **AWS** menu has the following options:

- **Credentials manager** – Select or manage one or more AWS profiles that provide credentials to access your AWS account. For more information, see Managing Cloud Canvas Profiles (p. 2223).

- **Cloud Canvas** – Choose from among the following options:
  
  - **Select a Deployment** – Specify the set of AWS resources for the project that you want Lumberyard Editor to work with. For more information, see Making a Cloud Canvas Deployment Active (p. 2355).
  
  - **Cloud Canvas Resource Manager** – Define and manage the AWS resources for your Lumberyard project. For a conceptual introduction to resource manager, see Understanding Cloud Canvas Resource Manager (p. 2110).
  
  - **Dynamic Content Manager** – Manage dynamic content updates for your game through AWS. For more information, see Using Dynamic Content Manager (p. 2155).

- **Commerce** – Choose from among the following options for monetizing your game:
  
  - **Merch by Amazon** – Create themed T-shirts for your game.
  
  - **Publishing on Amazon** – Distribute your Windows and macOS applications to customers.

- **GameLift** – Use the Amazon GameLift service to rapidly deploy and scale session-based multiplayer games with no upfront costs. For more information, see Amazon GameLift. The **GameLift** menu itself also has links to more information.

- **Open AWS Console** – Get quick access to the main AWS Management Console and to the consoles for Amazon Cognito, DynamoDB, Amazon S3, and Lambda:

These links use your currently active AWS profile to connect to AWS. You can use **Credentials Manager** to select which profile is active.
Managing Cloud Canvas Profiles

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Credentials Manager in Lumberyard Editor or the command line to manage one or more AWS profiles that provide the credentials required to access your AWS account.

The profile is saved locally on your machine in your AWS credentials file. This file is normally located in your `C:\Users<user name>\.aws` directory. The AWS Command Line Interface and the AWS Toolkit for Visual Studio can access these credentials.

Important
Do not share these credentials with anyone, and do not check them into source control. These grant control over your AWS account, and a malicious user could incur charges.

For more information, see AWS Security Credentials.

To open Credentials Manager

To open Credentials Manager, do one of the following:

- In Lumberyard Editor, click AWS, Credentials manager.
- In Cloud Canvas Resource Manager, click the name of the current profile in the Resource Manager toolbar:

You can use the Credentials Manager to select an existing AWS profile, edit an AWS profile, or add a new AWS profile.

To edit an existing AWS profile, click Edited selected profile. To add an AWS profile, click Add profile.

When adding or editing a profile, Lumberyard prompts you for the following:

Profile name – The name used for the profile.

AWS Secret Key – The AWS secret key needed to access the account.

AWS Access Key – The AWS access key needed to access the account.

To add your credentials by using the command line

1. Open a command line window and change to the root Lumberyard directory, which is the `dev` subdirectory of your Lumberyard installation directory (for example, `C:\lumberyard_version\dev`).
2. Enter the following at the command prompt, and then press Enter. Replace `<profile-name>` with a name of your choice (for example, `CloudCanvasAdminProfile`). Replace `<secret-key>` and `<access-key>` with the secret key and access key of your AWS account.

```
lmbr_aws profile add --profile <profile-name> --make-default --aws-secret-key <secret-key> --aws-access-key <access-key>
```

The `--make-default` option establishes the profile as your default profile for Cloud Canvas. The default profile eliminates the need to specify the profile each time you use Lumberyard Editor or run an `lmbr_aws` command.

**Using Resource Manager in Game Design**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Cloud Canvas Resource Manager to make deployments active, test mappings, use protected deployments, and to delete deployments and resources.

For information, see the following links in the Working with Deployments (p. 2352) section of the Cloud Canvas documentation.

- Making a Cloud Canvas Deployment Active (p. 2355)
- Testing Different Mappings (p. 2357)
- Using Protected Deployments (p. 2359)
- Deleting Cloud Canvas Deployments and Their Resources (p. 2360)

**Cloud Canvas Software Engineering Guide**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This section provides in-depth information for Cloud Canvas Resource Manager and the cloud gem framework.

**Topics**

- Resource Manager in Depth (p. 2224)
- Cloud Gem Framework (p. 2257)

**Resource Manager in Depth**

The effective use of Cloud Canvas Resource Manager is key to the success of any cloud-connected game project. This section provides in-depth information for software engineers.
For information on permissions in Resource Manager, see Understanding the Resource Manager Security System (p. 2370).

Topics
- Understanding Resource Status Descriptions (p. 2225)
- Editing Resource Manager Files (p. 2225)
- Working with JSON Files (p. 2227)
- Viewing the Cloud Canvas Progress Log (p. 2228)
- Working with Resource Groups (p. 2228)
- Resource Definitions (p. 2236)
- Permissions Metadata for Resource Definitions (p. 2240)
- Importing Resource Definitions into Cloud Canvas (p. 2242)
- Resource Deployments (p. 2247)
- Resource Mappings (p. 2250)
- Resource Manager Resource Group Parameters (p. 2251)
- Custom Resources (p. 2252)

Understanding Resource Status Descriptions

The status of AWS resources is displayed in the Cloud Canvas Resource Manager in places such as the progress log. The following list provides descriptions of common resource status codes. To see the reason for the current status, you can pause your mouse on the status text in the resource manager.

Create in progress – The resource is in the process of being created in AWS.
Create complete – The resource has been successfully created in AWS.
Create failed – The resource could not be created in AWS.
Update in progress – The resource is in the process of being updated in AWS.
Update complete – The resource was successfully updated in AWS.
Update failed – The resource could not be updated in AWS.
Delete in progress – The resource is in the process of being deleted in AWS.
Delete complete – The resource has been deleted in AWS.
Rollback in progress – An operation has failed and AWS CloudFormation is attempting to restore the resource to its previous state.
Rollback failed – A rollback has failed. The AWS resources in a CloudFormation stack that have this status are in an inconsistent state. You may have to delete and recreate the stack.

Editing Resource Manager Files

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
The navigation pane in the **Cloud Canvas Resource Manager** dialog contains a number of nodes that represent text files that are stored on disk. The resource-template.json (p. 2227) node is one example.

The child nodes of template files each represent one section of the parent node template file. These child nodes can help you locate and edit the resource definition sections of the parent node template file.

**Using the Internal Editor**

When you select a text file node in the navigation pane, the file content and text editing options are shown in the detail pane of **Cloud Canvas Resource Manager**. You can use the detail pane to view and edit the contents of the file. Use the **Edit, Search** menu item to search for text, and the **Previous** and **Next** buttons to navigate from one match to the next. After you have modified a file, you can save it by clicking **Save** in the toolbar or by choosing **File, Save**.

**Note**

The changes that you make in the template file child nodes are always saved to the parent node template file.

**Using an External Editor**

You can use an external script editor instead of the **Cloud Canvas Resource Manager** to edit files. You can specify which editor to use in Lumberyard Editor.

**To specify an external script editor**

- In Lumberyard Editor, click **File, Global Preferences, Editor Settings, General Settings, Files, External Editors, Scripts Editor**.

**To open a file in an external script editor**

- Right-click the file in the navigation pane and choose **Open in script editor**:

To copy the path of the template file to the clipboard, right-click the file in the navigation pane and choose **Copy path to clipboard**.

**Notes**

Note the following:

- Opening a child node of a template file in a script editor opens the full (parent) file for editing.
• If your project files are under source control, Lumberyard prompts you to check out files before they can be edited. The source control icon on the toolbar dynamically displays the status of a selected file in source control.
• If the contents of the file change on disk while there are unsaved changes in the editor, Lumberyard prompts you to load the updated contents from disk and replace the modified contents in the editor.

Working with JSON Files

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Some of the nodes in the Cloud Canvas Resource Manager navigation pane represent JSON template or settings files for your project. The content of these files is described in detail in Resource Definitions (p. 2236). When you select one of these nodes in the navigation pane, the detail pane shows the contents of that file. You can edit the file directly in the resource manager or use an external editor. For more information, see Editing Resource Manager Files (p. 2225).

In the navigation pane, some template file nodes have child nodes. Each of the child nodes represents one section of its parent node template file. These child nodes make it easier to find and edit the corresponding sections of the parent node template file. Any changes that you make in a child node are always saved in the corresponding section of the parent template file.

The following template is found in each resource group under the Resource Groups node:

resource-template.json

Each resource group has a resource-template.json node and a lambda-function-code child node. The resource-template.json file defines the group's resources. For more information, see resource-template.json (p. 2237). In the navigation pane, each of the nodes under resource-template.json represents one of the resources defined in a section of the resource-template.json file.

The following templates are found under the Administration (advanced) node:

project-settings.json

The project-settings.json file contains project configuration data. For more information, see project-settings.json (p. 2248).

project-template.json

The project-template.json file defines the resources used by Cloud Canvas Resource Manager. For more information, see project-template.json (p. 2238).

deployment-template.json

The deployment-template.json file defines the AWS CloudFormation stack resources for each of the projects resource groups. For more information, see deployment-template.json (p. 2238).

deployment-access-template.json

The deployment-access-template.json file defines the AWS CloudFormation stack resources that control access to each deployment's resources. For more information, see deployment-access-template.json (p. 2238) and Controlling Access to Resources (p. 2370).

user-settings.json

The user-settings.json file contains user specific settings. For more information, see user-settings.json (p. 2237).
Viewing the Cloud Canvas Progress Log

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Cloud Canvas Resource Manager Progress log shows the progress of AWS CloudFormation stack updates. During the update, the progress log expands from the bottom of the detail pane to display the progress of the update. You can adjust the amount of space the log uses by dragging the divider line between the panes.

To hide the progress log, drag the divider downward.

Working with Resource Groups

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Resource groups define the AWS resources that cloud-connected game features such as high score tables require. You define the resource groups locally and then upload them to AWS, where the features you specify are created in the cloud.

You can use the Cloud Canvas Resource Manager to manage resource groups collectively or individually. You can create a resource group locally, specify the AWS resources that you want the resource group to have, and then upload the resource group definition to have it created in AWS. This document shows you how to use Cloud Canvas Resource Manager to perform these and other resource group-related tasks.
Managing Resource Groups

To see the status of the resource groups that belong to your project's current deployment, click Resource Groups in the Cloud Canvas configuration navigation pane. Note the following:

- If you select Resource Groups and no AWS profile is configured, Lumberyard prompts you to provide one. A profile is required for Lumberyard to display the status of your project’s resources. For more information, see Managing Cloud Canvas Profiles (p. 2223).
- If you select Resource Groups and deployments exist but no deployment is active, Lumberyard prompts you to select one. For more information, see Making a Cloud Canvas Deployment Active (p. 2355).

The Resource Groups detail pane lists the resource groups in your current deployment:

![Resource Groups detail pane](image)

The Resource Groups detail pane has the following options:

**Upload all resources**

Starts the process of modifying your resources in AWS as needed to match all the definitions in all of your local resource groups. As the update proceeds, resource groups with the Pending status of Create change to the Status of Create complete. The update might take a few minutes.
Note the following:

- If you have not yet initialized your Lumberyard project with an AWS account for the resources that you want to upload, Lumberyard prompts you to do so. To prepare your Lumberyard project for use with AWS, you must be an administrator of the AWS account that you use. For more information, see Initializing Cloud Canvas Resource Manager (p. 2349).

- After you have initialized the project, Lumberyard prompts you to create a deployment for it. A deployment creates all the AWS resources that are specified in your resource group definition.

For more information, see Create Deployment (p. 2353).
Add resource group

Adds a new resource group definition to your Lumberyard project. A resource group definition represents a single game feature like a high score system. The definition specifies the AWS resources that the feature will use.

Clicking Add resource group opens the New resource group dialog:

Provide the following information:

- **Resource group name** – The name of the resource group. The name must be alphanumeric. Lumberyard uses this name to create an AWS CloudFormation stack resource definition in the deployment-template.json (p. 2238) file.

- **Example resources** – (Optional) Choose to include example resources in the resource group. You can study the examples to see how resources are defined in a resource group, or modify the examples to turn them into a feature for your project.

Resource group status

Shows the status of the AWS CloudFormation stack of each resource group in the active deployment. Resource group shows the resource group name. For descriptions of the remaining fields in this table, see Stack Resources Table (p. 2351) in the Working with Project Stacks (p. 2350) section.

Progress log

Shows the progress of AWS CloudFormation stack updates. For more information, see Viewing the Cloud Canvas Progress Log (p. 2228).

Managing Individual Resource Groups

Each child node of Resource Groups represents a resource group in your Lumberyard project. When you select one of these resource groups, the detail pane shows the status of the resource group.
Adding Resources to a New Resource Group

When you create a resource group, the group does not yet have any AWS resource definitions. Use the **Add resource** option to add one:

Importing Resources

You can also click **Import resource** to import an AWS resource that already exists in your AWS account. For more information, see Importing Resource Definitions into Cloud Canvas (p. 2242).

Creating Resources in AWS

The resource definitions that you add are created locally and only describe the AWS resources that you want to use. The resources themselves are not created in AWS until you click **Create resources**.
Individual Resource Group Status

You can use a resource group's status pane to manage the resource group.

In addition to Add resource and Import resource, the status pane for a resource group has the following options:

Upload Lambda code

Use this option to upload Lambda function code without updating the entire project stack.

Upload resources

After you have created one or more resource definitions, you click Upload resources to start the process of creating the resources in AWS. Resource creation follows the local resource definitions that you created with the Add resource option.
As the update proceeds, the resources with the Pending status of Create change to the Status of Create complete.

Remove resource group

Click Remove resource group to remove the selected resource group from your local configuration.

To delete the actual resources from AWS, use the Delete resources option as described in the section that follows.

Note
The remove resource operation does not remove the resource group's configuration data from the local disk. As long as that data exists on disk, you can restore the resource group by adding a new resource group with the same name.

Delete resources

The Delete resources option appears after you have removed a resource group from your local configuration (for example, by clicking Remove resource group) when the resources defined by the resource group still exist in AWS.
When you click **Delete resources**, Lumberyard prompts you to confirm the deletion of resources in AWS for the deployment that is currently active in Lumberyard Editor. You must select the **It is OK that this will permanently DELETE resources** check box before you can click **Yes**.

After you click **Yes**, the deletion operation might take several minutes to complete.

**Stack resources**

The **Stack resources** table shows the status of each of the AWS resources that you defined for the resource group. For descriptions of the fields in this table, see **Stack Resources Table** (p. 2351) in the **Working with Project Stacks** (p. 2350) section.

**resource-template.json**

For information about this node, see **Working with JSON Files** (p. 2227).

**lambda-function-code**

The **lambda-function-code** node and its child nodes correspond to the lambda-function-code directory in your project. The lambda-function-code directory contains the code that implements the AWS Lambda function resources that are defined by your resource group. For more information, see **The lambda-function-code Directory** (p. 2236). Also see related information for the **project-code** (p. 2235) node.

**project-code**

This node is located at the bottom of the **Administration (advanced)** section in the resource manager navigation tree. The project-code directory contains the code that implements the AWS Lambda
function resources that Cloud Canvas Resource Manager uses. For more information, see The project-code Directory (p. 2240). The project-code node contains file and directory child nodes. Click a file node to see or edit its contents in the detail pane. For more information, see Editing Resource Manager Files (p. 2225).

**Resource Definitions**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Resource definitions for your game project are AWS CloudFormation template files that determine the resources (for example, DynamoDB databases, Lambda functions, and access control information) that will be created in AWS for the game. Game code uses AWS resources and expect those resources to exist and to be configured in a specific way. The resource definitions determine this architecture and configuration.

**Game Resource Definitions**

A description of the resources required by the game is stored in files under the dev\{game\}\AWS directory, where dev is the Lumberyard installation \dev subdirectory and \{game\} is the directory identified by the sys_game_folder property in the \dev\bootstrap.cfg file. For example, if your game is the Samples Project, your resource definition path might be C:\lumberyard_version\dev \SamplesProject\AWS. These files should be checked into the project's source control system along with your other game code and data.

The default \{game\}\AWS directory contents are created by the lmbr_aws project create (p. 2405) command.

In addition, some user-specific configuration data is kept in the dev\Cache\{game\}\{OS\}\user\AWS directory. The contents of this directory should not be checked into the project's source control system.

The following shows the contents of these AWS directories.

```
dev\{game\}\AWS\resource-group\{resource-group-name\}lambda-function-code\(Lambda function Code)resource-template.jsonlocal-project-settings.json
dev\Cache\{game\}\{OS\}\user\AWS\user-settings.json
```

The .json files are described in the following sections.

**resource-group\{resource-group\} Directory**

The AWS resources used by the game are organized into separate resource groups. The resource-group directory contains these in individual \{resource-group\} subdirectories. Each \{resource-group\} subdirectory is typically named after your game project and can contain a lambda-function-code directory and a resource-template.json file.

**The lambda-function-code Directory**

The lambda-function-code subdirectory is present when a resource template defines Lambda function resources. This directory can contain source files that implement those functions. Lumberyard uploads the code from this directory when using the template to update the AWS CloudFormation stack.
Use of the lambda-function-code directory is deprecated. For more information, see Lambda Code Directories (p. 2240).

**resource-template.json**

A resource-template.json file is an AWS CloudFormation template that defines the AWS resources associated with each resource group. You can specify any AWS resource type supported by AWS CloudFormation in your resource-template.json file. For a list of the available resource types, see the AWS CloudFormation AWS Resource Types Reference.

To determine the resource group stacks to include in a deployment, Lumberyard reads the resource-template.json files in AWS\resource-group subdirectories and in the AWS subdirectories of enabled cloud gems. Following are two examples.

```
\dev\CloudGemSamples\AWS\resource-group\CloudGemAWSBehavior\resource-template.json
\dev\Gems\CloudGemMessageOfTheDay\AWS\resource-template.json
```

**local-project-settings.json**

The dev\*\AWS\local-project-settings.json file contains a ProjectStackId identifier that points to AWS, the Cloud Gem Framework version number, and a list of any resource groups that have been disabled for the project. The following example shows the format of the local-project-settings.json file for the Cloud Gem Samples project:

```
{
    "ProjectStackId": "arn:aws:cloudformation:{aws-region}:{aws-access-id}:stack/CloudGemSamples/{uuid}",
    "DisabledResourceGroups": [],
    "FrameworkVersion": "1.1.1"
}
```

Note
Prior to Lumberyard 1.11, the local-project-settings.json file kept a list of enabled resource groups. By default, all of a cloud gem's resource groups are enabled when the cloud gem is enabled. Listing only the disabled resource groups makes it easier to identify them for debugging. See resource-group disable (p. 2409).

Note
As of Lumberyard 1.8, the project-settings.json file is stored in the project's Configuration Bucket (p. 2248). The bucket is defined by the project's AWS CloudFormation stack template.

**ProjectStackId Property**

The ProjectStackId property identifies the AWS CloudFormation stack for the project. This stack contains the resources used by Cloud Canvas to manage your Lumberyard project. Initially, the ProjectStackId property is not present in the local-project-settings.json file. After the project stack is created, the ProjectStackId property is written to the file and is the project's AWS CloudFormation stack ID.

The ProjectStackId property is set by the project create (p. 2405) command. If for some reason you want to associate the project with an existing project stack, you can use the AWS Management Console to look up the stack's ARN and paste it into the local-project-settings.json file (navigate to AWS CloudFormation, select the stack, select Overview, and then copy the value of the Stack Id property).

**user-settings.json**

The user-settings.json file contains user-specific configuration data. The file is created on the first run of the Cloud Canvas Resource Manager or the first time that lmb CLAWS is run on a project.
File Location

The `user-settings.json` file is found at `dev\Cache\{game}\{OS}\user\AWS\user-settings.json`. It is not in the `dev\{game}\AWS` directory along with the other files described in this section because it should not be checked into the project's source control system.

Default AWS Profile

The `DefaultProfile` section of the `user-settings.json` file contains the AWS profile that the `lmbr_aws` command uses for the project. To set the profile for a Cloud Canvas project, use the following command:

```
lmbr_aws profile default --set profile_name
```

For more information, see profile default (p. 2404).

Project Template Files

Cloud Canvas project template files are [AWS CloudFormation templates](https://aws.amazon.com/cloudformation/) that define resources, stacks, and deployment permissions for a project. Following are the project template files:

- `project-template.json`
- `deployment-template.json`
- `deployment-access-template.json`

The content of each of these template files can be supplemented with a corresponding `*-extensions.json` file. Starting in Lumberyard 1.10, these files are located in the `\Gems\CloudGemFramework\vN\ResourceManager\resource_manager\templates` directory, where `N` represents a Cloud Gem framework version number.

`project-template.json`

The `project-template.json` file is an AWS CloudFormation template that defines project-wide resources that support the Cloud Canvas resource management system. For information about Cloud Canvas Resource Manager, see Understanding Cloud Canvas Resource Manager (p. 2110). For security-related information, see Understanding the Resource Manager Security System (p. 2370).

`deployment-template.json`

In versions of Lumberyard prior to 1.10, the project's copy of the `deployment-template.json` file contained stack definitions for each of the project's resource groups. These resources are now inserted into the base template before it is passed to AWS CloudFormation. To determine the resource group stacks to include, Lumberyard reads the `resource-template.json` files in `AWS\resource-group` subdirectories and in the `AWS` subdirectories of enabled gems.

`deployment-access-template.json`

The `deployment-access-template.json` file is an AWS CloudFormation Template that defines the resources used to secure a deployment. For information about Cloud Canvas Resource Manager, see Understanding Cloud Canvas Resource Manager (p. 2110). For security-related information, see Understanding the Resource Manager Security System (p. 2370).

Parameters

The deployment access stack defines parameters that identify the deployment and other resources that are needed to set up security for the deployment. A value for each of these parameters is provided by Cloud Canvas when a deployment is created.
**Resources**

This section describes some of the key resources that are defined in the example deployment-access-template.json file.

**Player**

The Player resource describes the IAM role that determines the access granted to the player. For example, for the game to invoke a Lambda function, the player must be allowed the lambda:InvokeFunction action on the Lambda function resource.

The role's policies are determined by the PlayerAccess metadata elements found on resources in the project's resource templates (see resource-template.json (p. 2237)). The role's policies are updated by the PlayerAccess custom resources that appear in the deployment-access-template.json (p. 2238) and in the resource-template.json (p. 2237) files. The PlayerAccessIdentityPool Amazon Cognito identity pool resource allows players to assume this role.

For more information, see PlayerAccessIdentityPool (p. 2239) and Controlling Access to Resources (p. 2370).

**PlayerLoginRole**

The PlayerLoginRole resource describes the IAM role that is temporarily assumed by the player as part of the login process.

For more information, see Controlling Access to Resources (p. 2370).

**PlayerLoginIdentityPool**

The PlayerLoginIdentityPool resource describes the Amazon Cognito identity pool that provides the player with a temporary identity during the login process.

For more information, see Controlling Access to Resources (p. 2370).

**PlayerAccessIdentityPool**

The PlayerAccessIdentityPool resource describes the Amazon Cognito identity pool that provides the player with a temporary identity during the login process.

For more information, see Controlling Access to Resources (p. 2370).

**Template Extension Files**

Starting in Lumberyard 1.10, you can use the following extension files to add resources to the project templates:

- deployment-access-template-extensions.json
- deployment-template-extensions.json
- project-template-extensions.json

You can use the extension files to define new outputs, metadata, and resources. The addition of template parameters is not supported.

To create extension files, you can use the lmbr_aws project create-extension-template (p. 2406) command. To create a template file of a particular type, use the corresponding argument, as the following examples show.

- To create a project-template-extensions.json file, type:

  ```bash
  lmbr-aws project create-extension-template --project
  ```
• To create a deployment-template-extensions.json file, type:

```
lmbr-aws project create-extension-template --deployment
```

• To create a deployment-access-template-extensions.json file, type:

```
lmbr-aws project create-extension-template --deployment-access
```

In the base (nonextension) template files, the Custom::AccessControl resource DependsOn property includes all the resources defined in the extension files if the DependsOn property does not specify AccessControl.

**Code Directories**

The location and naming of code directories has changed in Lumberyard 1.10. For more information, see [Cloud Gem Framework and Resource Manager Versioning (p. 2342)](#).

**Lambda Code Directories**

Starting in Lumberyard 1.10, we recommend that you put your Lambda code in `AWS\lambda-code\<lambda-name>` directories (for example, `dev\Gems\CloudGemPlayerAccount\AWS\lambda-code`) and your shared code in a `common-code` directory (for example, `dev\Gems\CloudGemPlayerAccount\AWS\common-code`).

In versions of Lumberyard prior to 1.10, the code for a resource group's Lambda functions was kept in an `AWS\lambda-function-code` directory. As of Lumberyard 1.10, the use of `AWS\lambda-function-code` and `AWS\<lambda-name>-lambda-code` directories is no longer recommended. Support for them will be removed in a future release. Instead, we recommend that you put your Lambda code in `AWS\lambda-code\<lambda-name>` directories.

In addition, the use of the `shared-lambda-code` directory is no longer recommended. Support for it will also be removed in a future release. Instead, use the more flexible `common-code` directory. For more information, see [Using Shared Code (p. 2342)](#). For information about upgrading your projects and cloud gems to version 1.0.0 of the Cloud Gem framework, see [Updating Projects and Cloud Gems to Version 1.0.0 of the Cloud Gem Framework (p. 2345)](#).

**The project-code Directory**

The `dev\Gems\CloudGemFramework\v\AWS\project-code` subdirectory contains the source code for the AWS CloudFormation Custom Resource handler that is used in the project's AWS CloudFormation templates. For information about custom resources, see [Custom Resources (p. 2252)](#).

It also contains the code that implements the token exchange step of the player login process. For more information, see [Controlling Access to Resources (p. 2370)](#).

**Permissions Metadata for Resource Definitions**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

To determine what permissions should go into a policy, the Custom::AccessControl resource looks for Cloud Canvas permissions metadata on resource definitions in resource group stacks. In the following example, the metadata on the Messages resource gives the SayHello AWS Lambda function permission to put items into a Amazon DynamoDB table. The metadata on the SayHello resource gives players permission to invoke the SayHello Lambda function.

```
...
```
"Messages": {
  "Type": "AWS::DynamoDB::Table",
  "Properties": {
    "AttributeDefinitions": [
      {
        "AttributeName": "PlayerId",
        "AttributeType": "S"
      }
    ],
    "KeySchema": [
      {
        "AttributeName": "PlayerId",
        "KeyType": "HASH"
      }
    ],
    "ProvisionedThroughput": {
      "ReadCapacityUnits": { "Ref": "ReadCapacityUnits" },
      "WriteCapacityUnits": { "Ref": "WriteCapacityUnits" }
    }
  },
  "Metadata": {
    "CloudCanvas": {
      "Permissions": [
        {
          "AbstractRole": "SayHello",
          "Action": "dynamodb:PutItem"
        }
      ]
    }
  }
},

"SayHello": {
  "Type": "AWS::Lambda::Function",
  "Properties": {
    "Description": "Example of a function called by the game to write data into a DynamoDB table.",
    "Handler": "main.say_hello",
    "Role": { "Fn::GetAtt": [ "SayHelloConfiguration", "Role" ] },
    "Runtime": { "Fn::GetAtt": [ "SayHelloConfiguration", "Runtime" ] },
    "Code": {
      "S3Bucket": { "Fn::GetAtt": [ "SayHelloConfiguration", "ConfigurationBucket" ] },
      "S3Key": { "Fn::GetAtt": [ "SayHelloConfiguration", "ConfigurationKey" ] }
    }
  },
  "Metadata": {
    "CloudCanvas": {
      "Permissions": [
        {
          "AbstractRole": "Player",
          "Action": "lambda:InvokeFunction"
        }
      ]
    }
  }
},
...

You can use the `lmbr_aws` command line tool to manage permissions metadata on the resource definitions in a resource group's `resource-template.json` file. For more information, see Permission Metadata Management (p. 2383).
Properties

Each Cloud Canvas Permission metadata object can have the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractRole</td>
<td>Required string or list of strings. Identifies the role whose policy will reflect the permission. For more information, see Role Mapping Metadata (p. 2377).</td>
</tr>
<tr>
<td>Action</td>
<td>Required string or list of strings. Identifies the actions to be placed in the policy. This is used as the Action property of a Statement in the policy. See AWS Service Actions and Condition Context Keys for Use in IAM Policies for a list of actions supported by IAM.</td>
</tr>
<tr>
<td>ResourceSuffix</td>
<td>Optional string or list of strings. Provides values that are appended to the ARN in the Resource property of a statement in the policy. There is one Resource property value for each suffix listed. If no suffixes are listed, the Resource property value is the resource ARN with no suffix.</td>
</tr>
</tbody>
</table>

See Also

For an overview of security in Cloud Canvas Resource Manager, see Understanding the Resource Manager Security System (p. 2370).

Importing Resource Definitions into Cloud Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Cloud Canvas resource importer to add definitions of existing AWS resources to a Cloud Canvas resource group. You can add resources by using the Cloud Canvas Resource Manager in Lumberyard Editor or at a command line prompt.

Importing Resources using Lumberyard Editor

In Lumberyard Editor, you can import a resource by specifying an Amazon Resource Name (ARN) or by choosing from a list.

To import a resource by using an ARN

1. From the Lumberyard Editor top menu, choose AWS, Cloud Canvas, Resource Manager.
2. In the navigation pane, select a resource group.
3. In the detail window, click Import resource, Import using ARN. You can also open the context (right-click) menu for the resource in the navigation pane and choose Import resource, Import using ARN.
4. In the Import using ARN dialog box, provide the ARN and name of the resource that you are going to import. Both are required.
After you have provided both items of information, the **Import** button is enabled.

5. **Import**.

**To import a resource by choosing from a list**

1. From the Lumberyard Editor top menu, choose **AWS, Cloud Canvas, Resource Manager**.
2. In the navigation pane, select a resource group.
3. In the detail window, choose **Import resource, Import using ARN**. You can also open the context (right-click) menu for the resource in the navigation pane and choose **Import resource, Import using ARN**.
4. In the **Import from list** dialog box, choose the AWS Region of the resource for **Region**. The default value is the region of the project stack if it exists. Resources start loading in the list as soon as you choose a region that has importable resources.
5. You can use the AWS service selector to filter the resources by service, and then use the **Search** box to filter resources by name.

6. Select the check box to the left of each resource that you want to import.

7. **Configure.**

8. In the **Configuration** dialog box, provide a reference name for each resource, or accept the default. The default name is the original name of the resource on AWS.

9. To delete a selected resource from the list, open the context (right-click) menu for the resource and choose **Delete.**

10. When you are ready, click **Import.** A progress bar displays. An **Import Error** message informs you of any errors that occur.
11. Click X to close the Import from list dialog box. The resources that you imported are listed in the details pane of Cloud Canvas Resource Manager.

**Importing Resource Definitions Using the Command Line**

To list and import resources using the Cloud Canvas command line, see resource-importer list-importable-resources (p. 2412) and resource-importer import-resource (p. 2411).

**Understanding Resource Definitions**

When you use the Cloud Canvas resource importer to import the definition of a resource, it is important to understand that you are importing the resource's definition, not the resource itself. For example, suppose you use the AWS console to create a high score table in DynamoDB called Table A. You create a game client that uploads scores, and send out the client to your players. Table A begins to populate with data from the players of your game.

You then decide to use Cloud Canvas to manage your resources and deployments. Using the Cloud Canvas Resource Manager, you import Table A because it has the exact configuration values that you want, and it has worked well for your use cases.

When you create a deployment with the imported resource, the deployment contains Table B, which is a new table with Table A's structure but not its data. Table B is managed by Cloud Canvas and has the same behavior as Table A. However, Table B is not a reference to Table A, and it does not have Table A's data or history. Keep this distinction in mind when you import resource definitions.

**Automatically Imported Resource Definitions**

Some of the existing resources that you select might be related to other resources. For example, Lambda functions can respond to events from the selected triggers. You can use event notifications from an Amazon S3 bucket to send alerts or trigger workflows. Cloud Canvas imports the related resources for you automatically.

Cloud Canvas uses the following naming conventions for automatically imported resource definitions.

<table>
<thead>
<tr>
<th>Source</th>
<th>Naming Convention</th>
<th>Example Name of Imported Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamoDB table, Lambda function, Amazon SNS topic, Amazon SQS queue</td>
<td><code>&lt;resource_name&gt;</code> + &quot;AutoAdded&quot; + <code>&lt;resource_type&gt;</code> + <code>&lt;counter&gt;</code></td>
<td>LambdaFunctionAutoAddedtable0</td>
</tr>
<tr>
<td>Lambda function configuration resource??</td>
<td><code>&lt;lambda_function_name&gt;</code> + &quot;Configuration&quot;</td>
<td>LambdaFunctionConfiguration</td>
</tr>
<tr>
<td>Lambda function policy resource</td>
<td><code>&lt;lambda_function_name&gt;</code> + &quot;Permission&quot;</td>
<td>LambdaFunctionPermission</td>
</tr>
<tr>
<td>DynamoDB table Lambda function event source</td>
<td><code>&lt;DynamoDB_table_name&gt;</code> + &quot;EventSource&quot;</td>
<td>DynamoTableEventSource</td>
</tr>
</tbody>
</table>
Resources Supported for Import

The following sections list the resource attributes and related resources that Cloud Canvas imports for each supported AWS service.

Dynamo DB Tables

For DynamoDB tables, Cloud Canvas imports the following resource attributes:

- AttributeDefinitions
- GlobalSecondaryIndexes
- KeySchema
- LocalSecondaryIndexes
- ProvisionedThroughput
- StreamSpecification

Amazon S3 Buckets

For Amazon S3 buckets, Cloud Canvas imports the following resource attributes:

- CorsConfiguration
- LifecycleConfiguration
- NotificationConfiguration
- Tags
- VersioningConfiguration
- WebsiteConfiguration

For Amazon S3 buckets, Cloud Canvas also imports the following related resources:

- Lambda functions
- Amazon SQS queues
- Amazon SNS topics

Lambda Functions

For Lambda functions, Cloud Canvas imports the following resource attributes:

- Code
- Description
- Handler
- MemorySize
- Role
- Runtime
- Timeout
- VpcConfig

For Lambda functions, Cloud Canvas also imports the following related resources:

- Lambda function configurations
• Lambda function permissions
• DynamoDB tables
• Event source mappings

**Amazon SNS Topics**

For Amazon SNS topics, Cloud Canvas imports the following resource attributes:

• DisplayName
• Subscription

For Amazon SNS topics, Cloud Canvas also imports any Lambda functions that are related resources.

**SQS Queues**

For SQS queues, Cloud Canvas imports the following resource attributes:

• DelaySeconds
• MaximumMessageSize
• MessageRetentionPeriod
• ReceiveMessageWaitTimeSeconds
• RedrivePolicy
• VisibilityTimeout

**Resource Deployments**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You implement deployments using AWS CloudFormation stacks. You create and manage the stacks using tools provided by Lumberyard.

A project may define any number of deployments, up to the limits imposed by AWS CloudFormation (for more information, see AWS CloudFormation Limits). Each deployment contains a completely independent set of the resources that the game requires. For example, you can have separate development, test, and release deployments so that your development and test teams can work independently of the deployment used for the released version of the game.

An AWS account that hosts a Lumberyard project contains the following resources:

• `{project}` – An AWS CloudFormation stack that acts as a container for all the project's deployments.
• `{project}`-Configuration – An S3 bucket used to store configuration data.
• `{project}`-ProjectResourceHandler – A Lambda function that implements the handler for the custom resources used in the templates. See Custom Resources (p. 2252).
• `{project}`-ProjectResourceHandlerExecution – An IAM role that grants the permissions used by the ProjectResourceHandler Lambda function when it is executing.
• `{project}`-PlayerAccessTokenExchange – A Lambda function that implements the token exchange step in the player login process. For more information, see Controlling Access to Resources (p. 2370).
• `{project}`-PlayerAccessTokenExchangeExecution– An IAM role that grants the permissions used by the PlayerAccessTokenExchange Lambda function when it runs.

• `{project}`-{deployment} – AWS CloudFormation stacks for each of the project's deployments.

• `{project}`-{deployment}Access – AWS CloudFormation stacks that control access to each of the project's deployments.

• `{project}`-{deployment}-PlayerAccessTokenExchangeIdentityPool – An Amazon Cognito identity pool used for player identity. For more information, see Controlling Access to Resources (p. 2370).

• `{project}`-{deployment}-PlayerLoginIdentityPool – An Amazon Cognito identity pool that provides the temporary player identity used during the player login process. For more information, see Controlling Access to Resources (p. 2370).

• `{project}`-{deployment}-{resource-group} – An AWS CloudFormation stack for each resource group of the project.

• `{project}`-{deployment}-{resource-group}-{resource} – The resources defined by a resource group. Because of how AWS CloudFormation works, parts of these names have unique identifiers appended to them. For example, for a project named MyGame with a deployment named Development and a feature named HighScore, the actual name of a Scores resource would be something like: MyGame-Development-1FLFSUKM3MC4B-HighScore-1T7DK9P46SQF8-Scores-1A1NIH66ZKPKRI. The tools provided by Lumberyard hide these actual resource names under most circumstances.

• `{project}`-{iam-policy-or-role} – An IAM policy or role that manages resource and deployment permissions. For more information, see Cloud Canvas Built-In Roles and Policies (p. 2373).

Configuration Bucket

The configuration Amazon S3 bucket is used to store configuration data for the project. The bucket is located at `/s3/buckets/<projectname>-configuration-<ID>/`. The tools provided with Cloud Canvas manage uploads to this bucket.

The upload directory contains objects that are uploaded from the `{game}/AWS` directory by the Cloud Canvas tools when stack management operations are performed. The uploads for each operation get assigned a unique `{upload-id}` value to prevent concurrent operations from impacting each other.

**project-settings.json**

The `project-settings.json` file contains project configuration data.

The structure of the `project-settings.json` file is as follows:

```json
{
    "key": "value",
    "deployment": {
        "deployment": {
            "key": "value",
            "resource-group": {
                "resource-group": {
                    "key": "value"
                }
            }
        }
    }
}
```

The `{key}` and `{value}` pairs represent individual settings. The pairs at the root apply to the project. The pairs under `{deployment}` apply to that deployment. The pairs under `{resource-group}` apply to that resource group. Either or both of `{deployment}` and `{resource-group}` can be *, to indicate
the settings they contain apply to all deployments or resource groups, respectively. Settings under a
named entry take precedence over settings under a * entry.

**Note**
As of Lumberyard 1.8, the `ProjectStackId` property is stored in the `dev\<project name>\AWS\local-project-settings.json` file. For more information, see ProjectStackId Property (p. 2237).

**DefaultDeployment Property**
The `DefaultDeployment` property identifies the deployment that is to be used by default when
working in Lumberyard Editor. The `DefaultDeployment` property in the `user-settings.json` file overrides this setting. The project and user defaults can be set using the `lmbr_aws deployment default` command. The DefaultDeployment setting is also used by the `lmbr_aws mappings update` command.

**ReleaseDeployment Property**
The `ReleaseDeployment` property identifies the deployment that is to be used in release builds of
the game. The `ReleaseDeployment` setting is used by the `lmbr_aws mappings update` command.

**DeploymentStackId Property, PendingDeploymentStackId**
The `DeploymentStackId` property identifies the AWS CloudFormation stack for a deployment. The `PendingDeploymentStackId` property identifies a stack whose deployment is pending. The project's resource groups are children of these stacks. For more information, see Resource Deployments (p. 2247).

The `DeploymentStackId` property is set by the `deployment create` command. If for some reason you want to associate the deployment with an existing deployment, you can use the AWS Management Console to look up the stack's ARN and paste it into the `project-settings.json` file (navigate to AWS CloudFormation, select the stack, select Overview, and then copy the value of the Stack Id property).

**DeploymentAccessStackId Property**
The `DeploymentAccessStackId` property identifies the AWS CloudFormation stack for the resources
that control access to a deployment.

The `DeploymentAccessStackId` is set by the `deployment create` command. If for some reason you want to associate the deployment with an existing deployment stack, you can use the AWS Management Console to look up the stack's ARN and paste it into the `project-settings.json` file (navigate to AWS CloudFormation, select the stack, select Overview, and then copy the value of the Stack Id property).

**parameter Property**
The `parameter` property provides the values for resource template parameters. The property must be in
the following format.

```
{
  ...
  "parameter": {
    "{template-parameter-name-1}": {template-parameter-value-1},
    ...
    "{template-parameter-name-n}": {template-parameter-value-n}
  }
  ...
}
```
Resource Mappings

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Resource mappings map the friendly names used in a game’s Resource Definitions (p. 2236) to the actual names of the resources created for one or more specific Resource Deployments (p. 2247). For example, a DynamoDB table name like LeaderboardTable would get mapped to a name like the following:

CloudGemSamples-CGSamplesDeployment-CloudGemLeaderboard-78AIXR0N0O4N-LeaderboardTable-1I1ZC6YO7KU7F

In the preceding example, CloudGemSamples is the name of the project stack, CGSamplesDeployment is the name of the deployment, and CloudGemLeaderboard is the name of the resource group. The 78AIXR0N0O4N and 1I1ZC6YO7KU7F parts of the resource name are inserted by AWS CloudFormation to guarantee that the resource name is unique over time. Thus, even if a resource is deleted and a new one with the same logical name is created, the physical resource ID will be different.

Usually different deployments, and consequently different mappings, are used for game development and for the released version of a game. Furthermore, different development, test, and other teams often work with their own deployments so that each team has distinct mappings.

The deployment used by default during development is specified in the project-settings.json (p. 2248) file. The file is located in the Amazon S3 configuration bucket at /s3/buckets/<projectname>-configuration-<ID>/project-settings.json. The file can be overridden for each user by the dev\Cache\{project\}pc\user\AWS\user-settings.json (p. 2237) file. You can change the default deployment by using the lmbr_aws deployment default (p. 2394) command or by using the Cloud Canvas Resource Manager (p. 2356).

The mappings that are used during development when the game is launched from the Lumberyard IDE by pressing Ctrl+G are stored in the user-settings.json (p. 2237) file. This file is updated automatically when the default deployment changes, when the default deployment is updated, and when Lumberyard Editor is started. To refresh it manually, you can use the lmbr_aws mappings update (p. 2402) command.

When a game launcher application created in Lumberyard launches a release build of a game, the mappings for the player and server roles are stored in the dev\{project\}Config\ directory. These mappings can be updated manually using the lmbr_aws mappings update --release (p. 2402) command. The command creates a <deployment_name>.player.awsLogicalMappings.json file for the player role and a <deployment_name>.server.awsLogicalMappings.json file for the server role. You can specify the deployment for the release mappings by using the --deployment {deployment_name} argument of the lmbr_aws mappings update command.

The Mappings Component

As of Lumberyard 1.11, the mapping functionality is implemented by a Mappings component within the Cloud Gem Framework Gem. The Mappings component (dev\Gems\CloudGemFramework \vN\Code\Source\MappingsComponent.h) is a required system component that replaces the former implementation in \dev\Code\CryEngine\LmbrAWS\Configuration\ClientManagerImpl. The Mappings component loads the <deployment_name>.player.awsLogicalMappings.json and <deployment_name>.server.awsLogicalMappings.json files from disk.

The Mappings Component EBus

The mappings component implements an EBus (p. 1851) found at dev\Code\CryEngine\CryCommon \CloudCanvas\CloudCanvasMappingsBus.h. When the Cloud Gem Framework makes calls to a
service API, it uses this EBus to translate the service name to the ARN. This EBus provides access to the mapping calls described in the following code.

```cpp
// Given the friendly name of the resource, return the ARN (physical name).
virtual AZStd::string GetLogicalToPhysicalResourceMapping(const AZStd::string& logicalResourceName) = 0;

// Set a friendly (logical) to ARN (physical) mapping for the specified resource type.
virtual void SetLogicalMapping(AZStd::string resourceType, AZStd::string logicalName, AZStd::string physicalName) = 0;

// Return all mappings that have the specified type (for example, AWS::Lambda::Function or Custom::ServiceApi).
virtual AZStd::vector<AZStd::string> GetMappingsOfType(const AZStd::string& resourceType) = 0;

// Return all mapping data.
virtual MappingData GetAllMappings() = 0;

// Load mappings from disk.
virtual bool LoadLogicalMappingsFromFile(const AZStd::string& mappingsFileName) = 0;

// If the protected flag is set in the mapping, the following function returns true.
// Because protected resources are likely to be live and customer facing, users should be
// warned accordingly and given the option not to connect.
virtual bool IsProtectedMapping() = 0;

// Set the mapping to protected.
virtual void SetProtectedMapping(bool isProtected) = 0;
```

In general, you should not have to access the low level mappings. Both ScriptCanvas and FlowNodes accept friendly names and translate them to ARNs before they make requests to AWS.

**Using Mappings in AWS Flow Nodes**

AWS flow nodes that define TableName (DynamoDB), FunctionName (Lambda), QueueName (Amazon SQS), TopicARN (Amazon SNS), or BucketName (Amazon S3) ports work with mappings. Set the port to a value like `{resource-group}.{resource}` where `{resource-group}` is the name of the resource group that defines the resource, and where `{resource}` is the name of the resource that appears in the Resources section of the resource group's resource-template.json file.

**Using Mappings in Lambda Functions**

Lambda function resources defined as part of a resource group often need to access other resources defined by that resource group. To do this, the function code needs a way to map a friendly resource name to the actual resource name used in AWS API calls. The LambdaConfiguration resource provides a way to such mappings, as well as other settings, to the lambda code. For more information, see LambdaConfiguration (p. 2256).

**Resource Manager Resource Group Parameters**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloud Canvas Resource Manager uses AWS CloudFormation templates to describe the AWS resources needed for a project. AWS CloudFormation templates can have parameters. You provide the values for these parameters when a template creates or updates a stack.
AWS CloudFormation template parameters are useful when you want separate deployments of a resource group to be configured differently. For example, you could configure an internal test instance of Amazon DynamoDB to use lower throughput than a public instance.

Follow the steps below to use AWS CloudFormation template parameters for your resource groups.

To use template parameters

1. Define a parameter in the AWS CloudFormation template. You must provide a default value for the parameter. Be sure not to remove or modify the parameter definitions used by Cloud Canvas (for example, `ConfigurationBucket` or `ConfigurationKey`).

2. Reference the parameter when you define a resource. You can use parameters to provide values for resource properties, including passing settings to Lambda functions through the `CloudCanvas::LambdaConfiguration (p. 2256)` resource.

3. Use the `lmbr_aws parameter list`, `parameter set`, and `parameter clear` commands to view and set parameter values. These commands are described later in this document.

4. Update (or create) the resource group stack using the `lmbr_aws resource-group upload` command, or click **Upload Resources** in the Cloud Canvas Resource Manager window in Lumberyard Editor.

Parameter Configuration

A project's parameter configuration is stored in a `project-settings.json (p. 2248)` file object in the project's configuration bucket. The bucket is defined by the project's AWS CloudFormation stack template.

You can specify parameter values for a specific deployment or for all deployments by using the `*` character as a wildcard in place of a deployment name. You can also specify parameter values for a specific resource group or for all resource groups by using the `*` in place of a resource group name. If you provide a parameter value for both a wildcard (`*`) entry and a named entry, the value from the named entry overrides the value from the wildcard entry.

The following table shows the `lmbr_aws` commands for listing, setting, and clearing parameter values. Your project must be initialized (that is, a project stack must have been created) before you can list, set, or clear parameter values. For usage details, visit the corresponding links in the table.

<table>
<thead>
<tr>
<th><code>lmbr_aws</code> command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parameter list</code> (p. 2403)</td>
<td>Lists the parameters currently configured for your project.</td>
</tr>
<tr>
<td><code>parameter set</code> (p. 2403)</td>
<td>Sets parameter configuration for your project.</td>
</tr>
<tr>
<td><code>parameter clear</code> (p. 2402)</td>
<td>Clears the specified parameter configuration for your project.</td>
</tr>
</tbody>
</table>

Custom Resources

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloud Canvas provides a number of **AWS CloudFormation custom resources** that can be used in the AWS CloudFormation template files for the project, deployment, and resource group. These custom resources are implemented by the Lambda function code found in the `dev\{game\}\AWS\project-code` directory and the `ProjectResourceHandler` resource defined in the `dev\{game\}\AWS\project-`
template.json file. Rather than static entities, these resources act more like library functions. Each custom resource has input and output properties.

A summary list of custom resources follows.

- **CognitoIdentityPool** *(p. 2253)* – Manages Amazon Cognito identity pool resources.
- **CognitoUserPool** *(p. 2254)* – Manages Amazon Cognito user pool resources.
- **EmptyDeployment** *(p. 2255)* – Used in the deployment-template.json when there are no resource groups defined.
- **LambdaConfiguration** *(p. 2256)* – Provides configuration data for Lambda function resources and maintains the Lambda function's execution role.
- **Helper** *(p. 2256)* – Provides convenience functions for use in templates.
- **Cloud Gem Framework Service API** *(p. 2305)* – Provides programmatic access to cloud gem functionality that you can use to implement cloud-connected features in your game.

For information on controlling access to resources, see Controlling Access to Resources *(p. 2370)*.

**CognitoIdentityPool**

The Custom::CognitoIdentityPool resource is used in the deployment-access-template.json file to create and configure Amazon Cognito identity pool resources.

**Input Properties**

- **ConfigurationBucket**
  
  Required. The name of the Amazon S3 bucket that contains the configuration data.

- **ConfigurationKey**
  
  Required. The Amazon S3 object key prefix where project configuration data is located in the configuration bucket. This property causes the custom resource handler to be executed by AWS CloudFormation for every operation.

- **IdentityPoolName**
  
  Required. The name of the identity pool.

- **UseAuthSettingsObject**
  
  Required. Must be either true or false. Determines whether the Amazon Cognito identity pool is configured to use the authentication providers that were created from the add-login-provider command.

- **AllowUnauthenticatedIdentities**
  
  Required. Must be either true or false. Determines whether the Amazon Cognito identity pool is configured to allow unauthenticated identities. See Identity Pools for more information on Amazon Cognito's support for authenticated and unauthenticated identities.

- **Roles**
  
  Optional. Determines the IAM role assumed by authenticated and unauthenticated users. See SetIdentityPoolRoles for a description of this property.

**Output Properties**

- **IdentityPoolName**
The name of the identity pool (same as the `IdentityPoolName` input property).

- `IdentityPoolId`

  The physical resource name of the identity pool.

**CognitoUserPool**

The `Custom::CognitoUserPool` resource is used in the [Player Account Cloud Gem (p. 2188)](resource-template.json) file to create and configure Amazon Cognito user pool resources. You can use this resource to add Amazon Cognito user pools and link them to an Amazon Cognito identity pool.

**Input Properties**

- `PoolName`
  
  Required. The name of the user pool.

- `ConfigurationKey`
  
  Required. The Amazon S3 object key prefix where project configuration data is located in the configuration bucket. This property causes the custom resource handler to be executed by AWS CloudFormation for every operation.

- `ServiceToken`
  
  Required. The handler for this custom resource type.

- `ClientApps`
  
  Required. A list of client apps to be created for the user pool. At least one must be provided.

- `LambdaConfig`
  
  Optional. A map of a Lambda trigger name to a Lambda function ARN. This map is passed to the Amazon Cognito `CreateUserPool` API when a user pool is created and to the Amazon Cognito `UpdateUserPool` API when a user pool is updated.

**Metadata: Linking a User Pool to an Identity Pool**

To link a `Custom::CognitoUserPool` to a `Custom::CognitoIdentityPool`, you add an Identities section to the Metadata CloudCanvas section, as in the following example.

```json
"Metadata": {
  "CloudCanvas": {
    "Identities": [
      {
        "IdentityPoolName": "PlayerAccess",
        "ClientApp": "DefaultClientApp"
      }
    ]
  }
}
```

The handler for `Custom::CognitoIdentityPool` looks for this metadata on `Custom::CognitoUserPool` resources when it configures the identity pool. The `IdentityPoolName` in the user pool's metadata must match the `IdentityPoolName` in the identity pool's properties. `ClientApp` must be one of the apps listed in the user pool's `ClientApps` property.

**Output Properties**

- `UserPoolName`
The name of the user pool.

- **UserPoolId**

  The ID of the user pool.

- **ClientApps**

  A list that contains the ClientId and ClientName of each of the user pool's client apps.

- **PhysicalResourceId**

  The physical resource ID of the user pool.

**EmptyDeployment**

The Custom::EmptyDeployment resource is used in the deployment-template.json file when there are no resource groups defined. This is necessary to satisfy the AWS CloudFormation requirement that a template define at least one resource.

This resource supports no input or output properties.

**ResourceGroupConfiguration**

The Custom::ResourceGroupConfiguration resource is used in the deployment-template.json to identify the location of the copy of the resource-template.json file in the configuration bucket that should be used for a specific resource group.

**Input Properties**

- **ConfigurationBucket**

  Required. The name of the Amazon S3 bucket that contains the configuration data.

- **ConfigurationKey**

  Required. The Amazon S3 object key prefix where the deployment configuration data is located in the configuration bucket.

- **ResourceGroup**

  Required. The name of the resource group that is to be configured.

**Output Properties**

- **ConfigurationBucket**

  The name of the Amazon S3 bucket that contains the configuration data. This is always the same as the ConfigurationBucket input property.

- **ConfigurationKey**

  The Amazon S3 object key prefix where the specified resource group's configuration data is located in the configuration bucket. This is the input ConfigurationKey with the string ResourceGroup and the value of ResourceGroup appended.

- **TemplateURL**

  The Amazon S3 URL of the resource group's copy of the resource-template.json in the configuration bucket. This value should be used as the resource group's TemplateURL property value.
**LambdaConfiguration**

The `Custom::LambdaConfiguration` resource is used in `resource-template.json` files to provide configuration data for Lambda function resources.

**Input Properties**

- **ConfigurationBucket**
  Required. The name of the Amazon S3 bucket that contains the configuration data.

- **ConfigurationKey**
  Required. The Amazon S3 object key prefix where configuration data for the resource group is located in the configuration bucket.

- **FunctionName**
  Required. The friendly name of the Lambda function resource being configured.

- **Settings**
  Optional. Values that are made available to the Lambda function code.

- **Runtime**
  Required. Identifies the runtime used for the Lambda function.

**Output Properties**

- **ConfigurationBucket**
  The name of the Amazon S3 bucket that contains the configuration data. This is always the same as the `ConfigurationBucket` input property.

- **ConfigurationKey**
  The Amazon S3 object key prefix where the specified function's zipped code is located in the configuration bucket.

- **Runtime**
  The Lambda runtime used by the function. This is always the same as the input `Runtime` property value.

- **Role**
  The ID of the Lambda function execution created for this function.

For information on how the `LambdaConfiguration` custom resource is used to allow Lambda functions to perform specified actions on specific project resources, see *Controlling Access to Resources* (p. 2370).

**Helper**

The `Custom::Helper` resource is used in templates to perform simple tasks similar to those performed by AWS CloudFormation's *Intrinsic Functions*.

**Input Properties**

- **Input**
  Provides a JSON object that is processed by the custom resource handler.
Output

The output is the processed version of the JSON object that is provided by the `Input` property.

Helper Function

The `Helper` resource supports the following function. The function is applied to the `Input` property value by resource handler to produce its output.

`HelperFn::LowerCase`

Converts a string to lowercase.

Syntax

```json
{ "HelperFn::LowerCase" : "converted-string-value" }
```

Example

The following example from the `project-template.json` file uses `HelperFn::LowerCase` to lower the case of a stack name.

```json
{
  ...
  "Resources": {
    ...
    "Helper": {
      "Type": "Custom::Helper",
      "Properties": {
        "Input": {
          "LowerCaseStackName": { "HelperFn::LowerCase": { "Ref": "AWS::StackName" } }
        },
        "ServiceToken": ...
      }
    },
    ...
    { "Fn::Join": [ "", [ "arn:aws:s3:::", { "Fn::GetAtt": [ "Helper", "LowerCaseStackName" ] }, "-*" ] ] }
    ...
  }
}
```

Cloud Gem Framework

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Lumberyard Cloud Gem Framework makes it easy to build popular cloud-connected features, such as dynamic content, leaderboards, and daily messages. The Cloud Gem Framework contains Cloud gems – Modular packages of discrete functionality and assets that include everything necessary for a game developer to include that functionality into their project, including backend and client functionality.

Cloud gems can be used out of the box in production, and they come with full source code in case you want to customize their behavior.

Topics

- Getting Started with the Cloud Gem Framework (p. 2258)
Lumberyard User Guide

Getting Started with the Cloud Gem Framework

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard cloud gems make it easy to create cloud-connected functionality for your games. You can use the cloud gems included with Lumberyard, or use Lumberyard's Cloud Gem Framework to create your own cloud gems.

Cloud Gems

Cloud gems are Lumberyard gems that use Cloud Canvas Resource Manager (p. 2110) (CCRM) to define AWS resources that support their operation. CCRM provides the infrastructure needed to create, configure, and access AWS resources. A cloud gem defines resources and implements the required operations to create a complete game subsystem.

Cloud gems use general purpose AWS features that operate in your AWS account. They are not game-specific services operated by Amazon. Cloud gems follow a serverless microservice architecture that makes use of AWS Lambda functions for computation, and Amazon DynamoDB, Amazon RDS, and Amazon S3 for data storage. However, a cloud gem can use any AWS service.

For example, a cloud gem could provide a leaderboard system that uses a DynamoDB table to store high scores. The gem could use an AWS Lambda function to validate submitted scores and allow fraudulent high scores to be deleted.

In this scenario, CCRM performs the following tasks:

• Creates the DynamoDB table and Lambda function defined by the gem.
• Enables the game client to invoke the API to submit a score.
• Protects the integrity of the leaderboard by removing fraudulent scores automatically.
• Permits the creation of a user interface that enables an authorized employee to remove fraudulent scores manually.

Cloud Gem Framework

The Cloud Gem Framework is a collection of tools and API operations that support the creation and operation of cloud gems.
The following diagram shows a high-level view of the Cloud Gem Framework architecture.

The Cloud Gem Framework contains the following major components:

- **Cloud Gem Framework Service API (p. 2305)** – An API that provides a secure interface to a cloud gem's functionality.
- **AWS API Jobs (p. 2286)** – Integrates the AWS API with the Lumberyard job execution systems (not illustrated).
- **HTTP Request Job (p. 2265)** – A Lumberyard job for executing HTTP requests (not illustrated).

### Creating a Cloud Gem

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes some cloud gems, such as leaderboards and dynamic content distribution, that provide full-featured services. You can use these gems in your games with a minimum of effort. Because the full source code for these cloud gems is included with Lumberyard, you can customize them or use them as examples for developing your own cloud gems.

You can develop your own cloud gems to create discrete packages of custom, cloud-connected functionality to enhance your game. Cloud gems leverage the power of AWS, benefit from the security model of AWS and Lumberyard, and can communicate with one another to create a seamless gaming experience.

You can easily create your own cloud gem by using the `lmbr_aws cloud-gem create` command.

### Prerequisites

1. Ensure that the Cloud Gem Framework gem is enabled for your project. Do one of the following:
   - Use the *Enabling Gems (p. 1064)* to enable the **Cloud Gem Framework** gem.
   
   OR
   
   - From the `\dev\Tools\LmbrSetup\Win` directory, enter the following command. Replace `<projectname>` with the name of your project.

```
lmbr gems enable <projectname> CloudGemFramework
```
2. Ensure that your project's Cloud Gem Framework resources correspond to the latest version of the Cloud Gem Framework gem. From a command prompt window on the \dev directory, enter the following command:

```bash
lmbr_aws project update-framework-version
```

3. If your cloud gem will use the Cloud Gem Framework Service API (p. 2305), do one of the following:
   - In Running Lumberyard Setup Assistant (p. 16), choose Compile the game code
   - OR
   - From a command prompt window on the \dev\Tools\LmbrSetup\Win directory, enter the following command:

```bash
lmbr capabilities enable compilegame
```

### To create a cloud gem

- From the \dev directory, enter the `lmbr_aws cloud-gem create` command. Use the following syntax:

```bash
lmbr_aws cloud-gem create --gem {gem-name} --initial-content {content-option} --enable
```

The syntax options are as follows.

- `--gem {gem-name}` – Specifies the name of the cloud gem.
- `--initial-content {content-option}` – Specifies the starter content for the cloud gem. For a list of content options, see `cloud-gem create` (p. 2392).
- `--enable` – Enables the cloud gem in your project.

### Example

The following command example creates a cloud gem called `MyCloudGem` that is enabled in the current project and has Amazon API Gateway and AWS Lambda function resources.

```bash
lmbr_aws cloud-gem create --gem MyCloudGem --initial-content api-lambda --enable
```

### Results of the Command

Depending on the parameter that you choose for the `--initial-content` option, the `cloud-gem create` command can do much of the initial cloud gem code work for you. For example, choosing the `api-lambda` parameter creates the following resources locally:

- A `\dev\Gems\{gem-name}\vN\ gem.json` file that declares a dependency on the Cloud Gem Framework.
- A `\dev\Gems\{gem-name}\vN\AWS` directory that includes the following items:
  - A `lambda-code` directory that contains skeleton `ServiceLambda` API code. `ServiceLambda` is an AWS Lambda function that implements your cloud gem's functionality.
  - A `swagger.json` file that describes the API for the cloud gem. For more information about the `swagger.json` file, see `Cloud Gem Framework Service API` (p. 2305).
  - A `resource-template.json` (p. 2237) file that defines the initial resources for the cloud gem. In the `api-lambda` example, the `resource-template.json` file specifies the following:
    - `ServiceApi`, `ServiceLambda`, and `ServiceLambdaConfiguration` resources.
• An output for the service API URL.
• An AccessControl resource configured as described in Controlling Access to Resources (p. 2370).

**Note**
Some gems have an obsolete cgp-resource-code directory.
• A `\dev\Gems\{gem-name\}\vN\Code\` directory with skeleton code that was automatically generated from the swagger.json file. The Code directory contains generated Using the Waf Build System (p. 63) files and the following subdirectories:
  • `\AWS\ServiceAPI\` – Contains `{gem-name}`ClientComponent.h and `{gem-name}`ClientComponent.cpp files. Use the skeleton code in these files to implement a Lumberyard component that you can use to invoke the cloud gem's service API operations from C++ or Lua.
  • `\Include\{gem-name\}\`– Contains a `{gem-name}`Bus.h EBus (p. 1851) file.
  • `\Source\`– Contains skeleton component, gem module and system component files.
  • `\Tests\`– Contains a skeleton `{gem-name}`Test.cpp file

**Developing your Cloud Gem**

After you use the `lmbr_aws cloud-gem create` command, you can further develop your cloud gem by doing the following:

• Add resource definitions that the cloud gem requires to the resource-template.json file. In Cloud Canvas Resource Manager, click Upload all resources to create and deploy the AWS resources that are defined in your resource-template.json file.
• Implement the cloud gem's functionality by adding code to the lambda-code directory.
• Add paths and operations to the swagger.json file that expose the functionality of your cloud gem. For more information, see Cloud Gem Framework Service API (p. 2305).
• Use Lumberyard components, EBus (p. 1851), and other features to implement game client functionality for the gem. Place the code in the cloud gem's `\dev\Gems\<gem-name>\vN\Code directory. You can use the generated service API client to access the functionality in the cloud gem's ServiceLambda Lambda function.

**Lambda Language Support in the Cloud Gem Framework**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

Lumberyard has support for Lambda deployment packages that include Lambda functions written in Node.js, Python 3.7 or later, Java, Go, or the .NET framework.

**To add a non-Python Lambda function to a cloud gem**

1. Add a new Lambda function and Lambda configuration to the Resources block of your resource_template.json file.

The following example shows the entries for the .NET Lambda function in the resource_template.json file of the AWS Lambda Language Demo cloud gem.

```json
[ ... ]
"Resources": {  
  "AccessControl": {  
```

Version 1.28
2261
"DependsOn": [
  "PythonLambda",
  "JavaJarLambda",
  "DotnetLambda",
  "GoLambda",
  "NodeLambda"
],
"Properties": {
  "ConfigurationBucket": {
    "Ref": "ConfigurationBucket"
  },
  "ConfigurationKey": {
    "Ref": "ConfigurationKey"
  },
  "ServiceToken": {
    "Ref": "ProjectResourceHandler"
  }
},
"Type": "Custom::AccessControl"
}

"DotnetLambda": {
  "Properties": {
    "Code": {
      "S3Bucket": {
        "Fn::GetAtt": [
          "DotnetLambdaConfiguration",
          "ConfigurationBucket"
        ]
      },
      "S3Key": {
        "Fn::GetAtt": [
          "DotnetLambdaConfiguration",
          "ConfigurationKey"
        ]
      }
    },
    "Handler": "DotnetLambda::DotnetLambda.Function::FunctionHandler",
    "Role": {
      "Fn::GetAtt": [
        "DotnetLambdaConfiguration",
        "Role"
      ]
    },
    "Environment": {
      "Variables": {
        "Fn::GetAtt": [
          "DotnetLambdaConfiguration",
          "CCSettings"
        ]
      }
    },
    "Runtime": {
      "Fn::GetAtt": [
        "DotnetLambdaConfiguration",
        "Runtime"
      ]
    }
  },
  "Type": "AWS::Lambda::Function"
},
"DotnetLambdaConfiguration": {
  "Properties": {
    "ConfigurationBucket": {
      "Ref": "ConfigurationBucket"
    }
  },
  "Version": "1.28"
"ConfigurationKey": {  
  "Ref": "ConfigurationKey"
},
  "FunctionName": "DotnetLambda",
  "Runtime": "dotnetcore1.0",
  "ServiceToken": {
    "Ref": "ProjectResourceHandler"
  }
},
  "Type": "Custom::LambdaConfiguration"
],
[ ... ]

2. Follow the instructions in Creating a Deployment Package to create a Lambda function deployment package for the runtime that you are using.

3. Use the FunctionName property of the function's LambdaConfiguration section to name the packaged .zip or .jar file.

4. Put the package file (for example, DotnetLambda.zip) in the lumberyard_version\dev\Gems \gem_directory\vN\AWS\lambda-code\ directory.

Notes

When you create non-Python Lambda functions for use with Cloud Canvas, note the following:

- The Visual Studio tools for .NET Lambda functions do not generate a Lambda function deployment package automatically. For steps, see .NET Core CLI in the AWS Lambda Developer Guide.
- When you use the Eclipse IDE and Apache Maven to build a Java .jar package, Maven generates a .jar file named project-SNAPSHOT.jar. The Cloud Canvas uploader supports this naming convention and the project.jar file.
- Each runtime has its own format for the Handler property. See the following table.

<table>
<thead>
<tr>
<th>Language</th>
<th>Handler Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET</td>
<td>Project::namespace.class::function</td>
</tr>
<tr>
<td>Java</td>
<td>package.class</td>
</tr>
<tr>
<td>Go</td>
<td>class must implement the RequestHandle class.</td>
</tr>
<tr>
<td></td>
<td>The name of the executable generated by the go build command.</td>
</tr>
</tbody>
</table>

Examples and Sample Level

The AWS Lambda Language Demo cloud gem included with Lumberyard contains skeleton "Hello World" Lambda function examples in Python, Java, .NET, Go, and node.js. You can find the relevant files in the lumberyard_version\dev\Gems\AWSLambdaLanguageDemo\v1\AWS\lambda-code directory.

Running the Sample Level

The CloudGemSamples project cloud gem includes a sample level named LambdaLanguage that shows Cloud Canvas support for Lambda languages.

Prerequisites

Running the sample requires the following prerequisites:
• You are using Lumberyard version 1.16 or later.
• Your Lumberyard project has the CloudGemSamples project enabled (in the Project Configurator (p. 43), choose CloudGemSamples).
• You used the Cloud Canvas Resource Manager (p. 2110) or the lmbr_aws command line tool (p. 2390) to create the following:
  • A project stack (p. 2350) for your project.
  • A deployment (p. 2352) for your project that includes the AWSLambdaLanguageDemo resource group.

To run the LambdaLanguage sample level

1. In Lumberyard Editor, choose File, Open or press Ctrl+O.
2. In the Open a Level dialog box, expand Levels.
3. Choose LambdaLanguage, and then click Open.
4. Click Play Game or press Ctrl+G to switch to gameplay mode. After a short pause, a message reports success, as seen in the following image.

<table>
<thead>
<tr>
<th>Sample Application</th>
<th>Resource Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DotNetLambda</td>
<td>AWS Toolkit for Visual Studio and .NET Core CLI</td>
</tr>
<tr>
<td>GoLambda</td>
<td>Creating a Deployment Package (Go)</td>
</tr>
<tr>
<td>JavaJarLambda</td>
<td>Creating a .jar Deployment Package Using Maven and Eclipse IDE (Java)</td>
</tr>
<tr>
<td>NodeLambda</td>
<td>Creating a Deployment Package (Node.js)</td>
</tr>
</tbody>
</table>

Additional Resources

For information on how the sample applications were built, see the following topics in the AWS Lambda Developer Guide.
Making HTTP Requests Using the Cloud Gem Framework

The Cloud Gem Framework and this documentation are in preview release and are subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Cloud Gem Framework Gem provides C++ classes and EBus interfaces to execute HTTP requests using the AZ::Job system. Your game client can use this feature to make HTTP requests for data from a public API such as Twitter or from a custom API. For example, your game could make HTTP requests to Twitter to see who is tweeting about your game.

To enable your game code to make HTTP requests

1. In the Project Configurator (p. 43), enable the Cloud Canvas Common and Cloud Gem Framework gems for your project.
2. In Lumberyard Editor, in Entity Inspector, click Add Component.
3. From the Cloud Gem Framework section, add the HttpClientComponent to an entity in your scene.
4. To make HTTP requests from your game code, perform one of the following steps:
   - From a Lua script component attached to your entity, add code based on the following example.
     ```lua
     local url = "https://my.url.com"
     local http_method = "GET"
     local json_body = "{}"
     ```
   - From C++, use Lumberyard’s EBus (p. 1851), as in the following example.
     ```cpp
     AZStd::string url = "https://my.url.com"
     AZStd::string httpMethod = "GET"
     AZStd::string jsonBody = "{}"
     EBUS_EVENT(HttpClientComponentRequestBus, MakeHttpRequest, url, httpMethod, jsonBody);
     ```
   - From C++, use HttpRequestJob, as in the following example.
     ```cpp
     AZStd::string url = "https://my.url.com"
     AZStd::string httpMethod = "GET"
     AZStd::string jsonBody = "{}"
     auto job = aznew HttpRequestJob(true, ServiceJob::GetDefaultConfig(),
     [this](int responseCode, AZStd::string content)
     {
       // handle success
     },
     [this](int responseCode)
     {
       // handle failure
     });
     job->SetUrl(url.c_str());
     job->SetHttpMethod(httpMethod);
     job->SetJsonBody(jsonBody.c_str());
     ```
Getting HTTP Responses Using Script

To get responses from a HTTP request, your script class needs an `HttpClientComponentNotificationBus` as in the following example.

```lua
function httpClientUsageExample:OnActivate()
    self.notificationHandler = HttpClientComponentNotificationBus.Connect(self, self.entityId);
end
```

Next, your script class must implement the `HttpClientComponentNotificationBus` functions `OnHttpRequestSuccess` and `OnHttpRequestFailure` as in the following example.

```lua
function myscript:OnHttpRequestSuccess(responseCode, responseBody)
    Debug.Log("HTTP RESPONSE -- " .. responseCode);
    Debug.Log("HTTP BODY -- " .. responseBody);
end

function myscript:OnHttpRequestFailure(errorCode)
    Debug.Log("HTTP Error-- " .. errorCode);
end
```

Getting HTTP Responses Using C++

To get the notifications in C++, you must create a component that inherits from `HttpClientComponentNotificationBus::Handler`. This class must implement `OnHttpRequestSuccess` and `OnHttpRequestFailure` and should be placed on the same entity as the `HttpClientComponent` in your level.

Cloud Gem Framework Resource Manager Hooks

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Cloud Canvas Resource Manager ensures that the AWS resources that the game uses match the definitions of those resources in the game's source code. However, you might need to add functionality to these processes. For example, for the Dynamic Content Cloud Gem, you must upload any default packages for the project immediately after you create a new deployment.

The resource manager provides some Python hook modules for cloud gems. The code for these modules is located in the `resource-manager-code` directory found in the cloud gem's AWS directory (for example, `\dev\Gems\CloudGemDynamicContent\AWS\resource-manager-code`).

The following modules are supported:

- `update.py` – Contains functions that are called before and after stack update operations. For more information, see Update Hook Functions (p. 2283).
- `command.py` – Contains functions that add new CLI and GUI support to the resource manager. For more information, see Command Hook Functions (p. 2284).

Hook Function Parameters

All hook functions are called with the following:
• A parameter that represents the hook function
• Some keyword arguments that are specific to each hook function

The parameter that represents the hook function is a `HookModule` object (p. 2275). The most important property provided by the `HookModule` object is `context`, which is a `Context Object` (p. 2267). The `Context` object gives your hook function access to the Cloud Canvas Resource Manager configuration data for the project.

**Futureproofing Your Functions**

When you write a hook function, always add Python's `**kwargs` construct as the last argument. That way your function can gracefully accept (and ignore) arguments that future versions of resource manager might add after your function is written.

For example, suppose resource manager passes `arg_a` and `arg_b` arguments to your hook function. You declare the hook function as follows:

```python
def a_hook(hook, arg_a, arg_b, **kwargs)
```

Later, when resource manager adds an `arg_c` argument, your hook function does not break. Your function still accepts `arg_a` and `arg_b` but can ignore `arg_c` because `arg_c` was passed through the `kwargs` parameter.

**Cloud Canvas Resource Manager Objects**

Cloud Canvas Resource Manager includes Python objects that you can use to access project configuration data and perform various project operations.

**Note**
The source code for these objects is located in the Lumberyard dev\Tools\lmbr_aws \AWSResourceManager directory.

**Context Object**

The `Context` object has properties that provide access to project configuration and that perform operations such as adding resource definitions to a resource group template.

You can get a `Context` object instance from the `context` property of the `HookModule` object instance passed to hook functions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws</td>
<td>An AWSContext object.</td>
</tr>
<tr>
<td>config</td>
<td>A ConfigContext object.</td>
</tr>
<tr>
<td>gem</td>
<td>A GemContext object.</td>
</tr>
<tr>
<td>hooks</td>
<td>A HookContext object.</td>
</tr>
<tr>
<td>resource_groups</td>
<td>A ResourceGroupContext object.</td>
</tr>
<tr>
<td>stack</td>
<td>A StackContext object.</td>
</tr>
<tr>
<td>view</td>
<td>A ViewContext object.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.
AWSContext Object

The AWSContext object has helper functions and properties related to AWS clients and credentials.

You can get an AWSContext object instance from the aws property of a Context object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assume_role(logical_role_id, deployment_name)</td>
<td>Assumes a role that uses the current AWS credentials. New clients that were created from the client function use the temporary credentials that were created by assuming the role. These credentials have a 3600 second (5 minute) lifetime.</td>
</tr>
<tr>
<td></td>
<td>The logical_role_id parameter can be the logical name of a AWS::IAM::Role resource defined in the project-template.json or deployment-access-template.json files. In the case of a role that is defined in the deployment-access-template.json file, the deployment_name identifies the actual role that will be assumed.</td>
</tr>
<tr>
<td>client(service_name, region=None, use_role=True)</td>
<td>Creates a Python AWS API (boto3) client for the specified AWS service. If use_role is True, the client uses the credentials that are created when assume_role was called.</td>
</tr>
<tr>
<td></td>
<td>The returned object is a wrapper around the actual boto3 client object. The wrapper logs all API call attempts and responses when the --verbose option is passed to lmbr_aws. This aids in debugging.</td>
</tr>
<tr>
<td></td>
<td>The client wrapper also performs retries with exponential back off.</td>
</tr>
<tr>
<td>get_credentials_file_path()</td>
<td>Returns the full path to the .aws/credentials file.</td>
</tr>
<tr>
<td>get_default_profile()</td>
<td>Gets the AWS credentials profile name that is used by default when clients are created.</td>
</tr>
<tr>
<td>get_temporary_credentials(logical_role_id, deployment_name, duration_seconds)</td>
<td>Creates temporary credentials by using the current credentials to assume a role.</td>
</tr>
<tr>
<td></td>
<td>The logical_role_id parameter can be the logical name of a AWS::IAM::Role resource defined in the project-template.json or deployment-access-template.json files. In the case of a role defined in the deployment-access-template.json file, the deployment_name identifies the actual role that will be assumed.</td>
</tr>
<tr>
<td></td>
<td>The return value is a Python dict with the following properties:</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AccessKeyId</td>
<td>Returns an AWSCredentials object that contains the data read from the .aws/credentials file.</td>
</tr>
<tr>
<td>SecretAccessKey</td>
<td></td>
</tr>
<tr>
<td>SessionToken</td>
<td></td>
</tr>
<tr>
<td>load_credentials()</td>
<td></td>
</tr>
<tr>
<td>profile_exists(profile_name)</td>
<td>Determines whether the specified profile exists in the .aws/credentials file.</td>
</tr>
<tr>
<td>save_credentials(credentials)</td>
<td>Saves a modified AWSCredentials object into the .aws/credentials file.</td>
</tr>
<tr>
<td>session</td>
<td>The boto3 session object that is used by client().</td>
</tr>
<tr>
<td>set_default_profile</td>
<td>Sets the AWS credentials profile that is used when clients are created. This resets the session property.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.

**AWSCredentials Object**

The AWSCredentials object contains AWS credential information that is read from the .aws/credentials file. This is essentially a wrapper around a Python ConfigParser object. The AWSCredentials object handles the AWS credential file's use of the default section, which conflicts with how ConfigParser handles defaults.

To get an AWSCredentials object instance, use the load_credentials function on an AWSContext object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_section(section_name)</td>
<td>Adds a section.</td>
</tr>
<tr>
<td>get(section_name, option_name)</td>
<td>Gets the value of an option in a section.</td>
</tr>
<tr>
<td>has_option(section_name, option_name)</td>
<td>Determines whether an option exists in a section.</td>
</tr>
<tr>
<td>has_section(section_name)</td>
<td>Determines whether a section exists.</td>
</tr>
<tr>
<td>items(section_name)</td>
<td>Returns a list of name value pairs of the options in a section.</td>
</tr>
<tr>
<td>options(section_name)</td>
<td>Returns a list of options in a section.</td>
</tr>
<tr>
<td>read(path)</td>
<td>Reads credentials from a file with the specified path.</td>
</tr>
<tr>
<td>remove_option(section_name, option_name)</td>
<td>Removes an option from a section.</td>
</tr>
<tr>
<td>remove_section(section_name)</td>
<td>Removes a section.</td>
</tr>
</tbody>
</table>
### Name | Description
---|---
sections() | Returns a list of section names. Each profile is represented by a section.
set(section_name, option_name, value) | Sets an option value.
write(path) | Writes credentials to a file with the specified path.

Other properties or functions of this object are internal to resource manager and should not be used.

**ConfigContext Object**

The ConfigContext object has properties and functions that provide access to project and deployment configuration data.

You can get a ConfigContext object instance from the `config` property of a Context object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws_directory_path</td>
<td>The full path to the currently enabled game project (as determined by the <code>bootstrap.cfg</code> file or options passed to <code>lmbr_aws</code>).</td>
</tr>
<tr>
<td>base_resource_group_directory_path</td>
<td>The directory where resource groups that are defined by the project (as opposed to those defined by gems) are located. This is typically the <code>resource-group</code> subdirectory that is found in the directory identified by <code>aws_directory_path</code>.</td>
</tr>
<tr>
<td>clear_user_default_deployment()</td>
<td>Clears the default deployment setting.</td>
</tr>
<tr>
<td>clear_user_default_profile()</td>
<td>Clears the default AWS credentials profile name. This does not update the <code>.aws/credentials</code> file. The name of the default profile is saved in the Cloud Canvas Resource Manager's user settings file.</td>
</tr>
<tr>
<td>configuration_bucket_name</td>
<td>The name of the project's Configuration Amazon S3 bucket.</td>
</tr>
<tr>
<td>copy_default_lambda_function_content(destination_path)</td>
<td>Copies the default Lambda function content to the specified location.</td>
</tr>
<tr>
<td>copy_default_project_content(destination_path)</td>
<td>Copies the content from the default project AWS directory to the specified location.</td>
</tr>
<tr>
<td>copy_default_resource_group_content(destination_path)</td>
<td>Copies the content from the default resource group to the specified location.</td>
</tr>
<tr>
<td>copy_example_resource_group_content(destination_path)</td>
<td>Copies the content from the example resource group to the specified location.</td>
</tr>
</tbody>
</table>
| default_deployment | The name of the deployment that is the default for the current user if a default name has been specified. If a default has been specified, the value is that of Version 1.28

2270
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_default_deployment</td>
<td>user_default_deployment; otherwise, it is that of project_default_deployment.</td>
</tr>
<tr>
<td>deployment_access_template</td>
<td>A Python object that contains the content of the deployment access template file.</td>
</tr>
<tr>
<td>deployment_access_template_path</td>
<td>The full path to the project's deployment-access-template.json file.</td>
</tr>
<tr>
<td>deployment_names</td>
<td>A list that contains the names of the project's deployments.</td>
</tr>
<tr>
<td>deployment_stack_exists</td>
<td>Determines whether a deployment stack exists.</td>
</tr>
<tr>
<td>finalize_deployment_stack_ids</td>
<td>Commits the pending deployment and deployment access stack IDs as the deployment's actual stack IDs. See set_pending_deployment_stack_id and set_pending_deployment_access_stack_id.</td>
</tr>
<tr>
<td>game_directory_path</td>
<td>The full path to the currently enabled game project (as determined by the bootstrap.cfg file or the options that are passed to lmbr_aws).</td>
</tr>
<tr>
<td>gem_directory_path</td>
<td>The full path to the directory where gems are defined. This is the Gems subdirectory of the directory identified by root_directory_path.</td>
</tr>
<tr>
<td>get_default_deployment_stack_name</td>
<td>Gets the default stack name for a deployment with the specified name.</td>
</tr>
<tr>
<td>get_deployment_access_stack_id</td>
<td>Gets the ARN of a deployment access stack. If the specified deployment stack doesn't exist, raises a HandledError or returns None if optional is True.</td>
</tr>
<tr>
<td>get_deployment_stack_id</td>
<td>Gets the ARN of a deployment stack. If the specified deployment stack doesn't exist, raises a HandledError or returns None if optional is True.</td>
</tr>
<tr>
<td>get_game_directory_name</td>
<td>Gets the name of the game directory from the bootstrap.cfg file.</td>
</tr>
<tr>
<td>get_project_stack_name</td>
<td>Gets the name of the project stack.</td>
</tr>
<tr>
<td>get_protected_deployment_names</td>
<td>Returns a list of the protected deployment names. For more information, see protect_deployment and unprotected_deployment.</td>
</tr>
<tr>
<td>get_resource_group_stack_id</td>
<td>Gets the ARN of a resource group stack. If the specified resource group stack doesn't exist, raises a HandledError or returns None if optional is True.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>local_project_settings_path</td>
<td>Full path to the file that stores project's settings that are kept in a source-controlled file (as opposed to project settings stored in AWS). Typically, this is the local-project-settings.json file in the directory that is identified by aws_directory_path.</td>
</tr>
<tr>
<td>no_prompt</td>
<td>When True, specifies that the --no-prompt option was passed to lmbr_aws. This option is typically used when calling lmbr_aws from scripts. When this option is set, the hook function should raise an error instead of prompting the user.</td>
</tr>
<tr>
<td>project_code_path</td>
<td>The path to the project's project-code directory.</td>
</tr>
<tr>
<td>project_default_deployment</td>
<td>The name of the deployment that has been designated as the default for the project, if any. See also user_default_deployment and default_deployment.</td>
</tr>
<tr>
<td>project_initialized</td>
<td>True if the project has been initialized and project_stack_id has a value; False otherwise.</td>
</tr>
<tr>
<td>project_resource_handler_id</td>
<td>The ARN of the ProjectResourceHandler Lambda function. Cloud Canvas–defined resource templates require that this value be provided as the ProjectResourceHandler parameter value.</td>
</tr>
<tr>
<td>project_settings</td>
<td>A ProjectSettings object that contains project settings that are loaded from AWS (as opposed to the content of local_project_settings).</td>
</tr>
<tr>
<td>project_stack_id</td>
<td>The project stack ID, if the project stack has been created.</td>
</tr>
<tr>
<td>project_template</td>
<td>A Python object that contains the content of the project template file.</td>
</tr>
<tr>
<td>project_template_path</td>
<td>The full path to the project's project-template.json file.</td>
</tr>
<tr>
<td>protect_deployment(deployment_name)</td>
<td>Marks a deployment as protected. It requires special confirmation before you can connect debug builds of the game to protected deployments. See also unprotected_deployment and get_protected_deployment_names.</td>
</tr>
<tr>
<td>refresh_user_settings()</td>
<td>A function that reloads the user_settings object from the user settings file.</td>
</tr>
<tr>
<td>release_deployment</td>
<td>The name of the deployment that has been designed as the release deployment of the project.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>remove_deployment(deployment_name)</td>
<td>Removes a deployment from the project's configuration.</td>
</tr>
<tr>
<td>root_directory_path</td>
<td>The full path to the Lumberyard root directory (the \dev directory), which contains project-specific directories.</td>
</tr>
<tr>
<td>save_deployment_access_template()</td>
<td>Saves the current deployment_access_template object to the project's deployment-access-template.json file.</td>
</tr>
<tr>
<td>save_deployment_template()</td>
<td>Saves the current deployment_template object to the project's deployment-template.json file.</td>
</tr>
<tr>
<td>save_project_template()</td>
<td>A function that saves the current project_template object to the project's project-template.json file.</td>
</tr>
<tr>
<td>set_pending_deployment_access_stack_id(deployment_access_stack_id)</td>
<td>Sets the ID for a deployment access stack that is being created. For more information, see finalize_deployment_stack_ids and set_pending_deployment_stack_id.</td>
</tr>
<tr>
<td>set_pending_deployment_stack_id(deployment_stack_id)</td>
<td>Sets the ID for a deployment stack that is being created. For more information, see finalize_deployment_stack_ids and set_pending_deployment_access_stack_id.</td>
</tr>
<tr>
<td>set_project_default_deployment(deployment_name)</td>
<td>Sets the project's default deployment for the project. Overridden by the value for user's default deployment if one is specified.</td>
</tr>
<tr>
<td>set_release_deployment(deployment_name)</td>
<td>Sets the release deployment for the project.</td>
</tr>
<tr>
<td>set_user_default_deployment(deployment_name)</td>
<td>Sets the default deployment for the user. After it is set, overrides the project default deployment.</td>
</tr>
<tr>
<td>set_user_default_profile(profile_name)</td>
<td>Sets the user's default AWS credentials profile. This does not update the user's .aws/credentials file; the name of the default profile is saved in the Cloud Canvas Resource Manager's user settings file.</td>
</tr>
<tr>
<td>unprotectedeployment(deployment_name)</td>
<td>Reverses the action of protect_deployment.</td>
</tr>
<tr>
<td>user_default_deployment</td>
<td>The name of the deployment that has been designated as the default by the current user. For more information, see project_default_deployment and default_deployment.</td>
</tr>
<tr>
<td>user_default_profile</td>
<td>The user's default AWS profile name as read from user settings.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
user_directory_path | The full path to the user directory in the Lumberyard cache directory. This is where user-specific configuration is stored.
user_settings | A Python object that contains the contents of the user settings file.
user_settings_path | The full path to the JSON format text file that resource manager uses to store user-specific configuration. This configuration includes default deployment and AWS credentials profile names. This path is typically the `user-settings.json` file in the directory specified by `user_directory_path`.

Other properties or functions of this object are internal to resource manager and should not be used.

**Gem Object**

The Gem object provides access to gem-specific configuration data. To get Gem object instances, use the `enabled_gems` property of a GemContext object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws_directory</td>
<td>The full path to the gem's AWS directory.</td>
</tr>
<tr>
<td>aws_directory_exists</td>
<td>True if the gem's AWS directory exists.</td>
</tr>
<tr>
<td>display_name</td>
<td>The gem display name.</td>
</tr>
<tr>
<td>file_object</td>
<td>A Python object that contains the contents of the gem's <code>gem.json</code> file.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the gem.</td>
</tr>
<tr>
<td>project_resources</td>
<td>Project resource definitions that are inserted into the project's stack template before the template is uploaded. These definitions are read from a <code>project-template.json</code> file in the gem's AWS directory.</td>
</tr>
<tr>
<td>resource_group_name</td>
<td>The gem's resource group name, if any. A gem has a resource group name only if it has a <code>resource-template.json</code> file.</td>
</tr>
<tr>
<td>root_directory</td>
<td>The full path to the gem's directory.</td>
</tr>
<tr>
<td>uuid</td>
<td>The unique ID of the gem.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to the resource manager and should not be used.

**GemContext Object**

The properties and methods of the GemContext object provide access to the project's gem configuration.
You can get a GemContext object instance using the gems property of a Context object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled_gems</td>
<td>A list containing a gem object for each gem that is enabled for the project.</td>
</tr>
<tr>
<td>get_by_resource_group_name(resource_group_name)</td>
<td>Returns the gem object associated with the specified resource group, or None if the resource group doesn't exist or isn't associated with a gem. By default the resource group name is the same as the gem name, but a ResourceGroupAlias property can be put in the gem's gem.json file to override this default.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.

**HandledError Object**

A HandledError object is Python Exception object. You can use it in a hook function to cause an expected error to be displayed without producing a stack trace. In general, the resource manager considers other kinds of exceptions to be unexpected errors and might show additional debug information that users should not see for expected errors.

The HandledError class is defined in the AWSResourceManager.errors module (\dev\Tools \lmbr_aws\AWSResourceManager\errors.py). To create an instance, see the following example.

```python
from AWSResourceManager.errors import HandledError

def my_function():
    if something_is_wrong:
        raise HandledError('Something is wrong.')
```

**HookModule Object**

The first argument (and the only positional argument) passed to a hook function is a HookModule object that represents the hook module itself. This object's properties give you access to project configuration data.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Provides access to a Context object.</td>
</tr>
<tr>
<td>hook_name</td>
<td>The name of the cloud gem that defines the hook module.</td>
</tr>
<tr>
<td>hook_path</td>
<td>The full path to the cloud gem's AWS directory where the hook module is defined.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.

**HookContext Object**

The HookContext object provides functionality for working with Cloud Canvas Resource Manager hooks.
You can get a HookContext object instance from the hooks property of the Context object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_module_handlers(module_name, handler_name, args=(), kwargs={}, deprecated=False)</td>
<td>Calls the hook function identified by handler_name in the module identified by module_name. Use the kwargs parameter to pass keyword arguments to the handler function. The args parameter can only be used if deprecated is True. If deprecated is True, a warning regarding the deprecated function is displayed.</td>
</tr>
<tr>
<td>call_single_module_handler(module_name, handler_name, defining_module_name, args=(), kwargs={}, deprecated=False)</td>
<td>Calls a hook function defined by the gem or resource group that is specified by the defining_module_name parameter. For a description of the other parameters, see call_module_handlers.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.

**ProjectSettings Object**

The ProjectSettings object manages the project configuration data that is stored in the project-settings.json file in the project's Amazon S3 Configuration bucket.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_deployment(deployment_name)</td>
<td>Returns the configuration dictionary for the specified deployment. The dictionary contains the following keys:</td>
</tr>
<tr>
<td></td>
<td>DeploymentStackId – The ARN of the deployment stack.</td>
</tr>
<tr>
<td></td>
<td>DeploymentAccessStackId – The ARN of the deployment access stack.</td>
</tr>
<tr>
<td></td>
<td>resource-group – The resource group configuration dictionary for the deployment. For a description of this value, see get_resource_group_settings.</td>
</tr>
<tr>
<td>get_deployments()</td>
<td>Returns the entire configuration dictionary for deployments. The deployment name is the key. The special deployment name *, which contains settings for all deployments, can be present. For a description of the value dictionary, see get_deployment.</td>
</tr>
<tr>
<td>get_project_default_deployment()</td>
<td>Returns the default deployment for the project.</td>
</tr>
<tr>
<td>get_release_deployment()</td>
<td>Returns the release deployment for the project.</td>
</tr>
<tr>
<td>get_resource_group_settings(deployment_name)</td>
<td>Returns a dictionary containing the resource group configuration for the specified deployment. The resource group name is the key. The special resource group name *, which contains settings for all resource groups, can be present.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>remove_deployment(deployment_name)</td>
<td>Removes the configuration for the specified deployment name. To commit this change, call save.</td>
</tr>
<tr>
<td>remove_project_default_deployment()</td>
<td>Clears the default deployment for the project. To commit this change, call save.</td>
</tr>
<tr>
<td>remove_release_deployment()</td>
<td>Clears the release deployment for the project. To commit this change, call save.</td>
</tr>
<tr>
<td>save()</td>
<td>Saves the current configuration to the Amazon S3 bucket.</td>
</tr>
<tr>
<td>set_project_default_deployment(deployment_name)</td>
<td>Sets the default deployment for the project. To commit this change, call save.</td>
</tr>
<tr>
<td>set_release_deployment(deployment_name)</td>
<td>Sets the release deployment for the project. To commit this change, call save.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.

**ResourceGroup Object**

The ResourceGroup object encapsulates a resource group's configuration. If multiple resource group stacks for a given resource group exist, each resource group stack is associated with a single deployment.

To get ResourceGroup object instances, use a ResourceGroupContext object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_output(logical_id, description, value, force=False)</td>
<td>Adds an output value to the resource group's resource template. If force is False, an existing output that has the same name is not replaced but is replaced if force is True. To save the changes, call save_template. Returns True if outputs are added.</td>
</tr>
<tr>
<td>add_parameters(parameter_definitions, force=False)</td>
<td>Adds parameter definitions to the resource group's resource template. The parameter_definitions parameter is a dictionary that is merged into the parameter definitions in the template. If force is False, existing definitions are not replaced but are replaced if force is True. To save the changes, call save_template. Returns True if parameter definitions are added.</td>
</tr>
<tr>
<td>add_resources(resource_definitions, force=False, dependencies=None)</td>
<td>Adds resource definitions to the resource group's resource template. The resource_definitions parameter is a dictionary that is merged into the template's current Resources object value. If</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
force | If `force` is `False`, existing definitions are not be replaced but are replaced if `force` is `True`. The `dependencies` parameter can be a dictionary that specifies a list of values that are added to the existing resources `DependsOn` list. New resources are always added to the `AccessControl` resource's `DependsOn` list even if the `dependencies` parameter is not specified. To save the changes, call `save_template`. Returns `True` if resource definitions are added.

copy_directory(source_path, relative_destination_path='.', force=False) | Copies the contents of a specified directory into the resource group's directory. If a value for `relative_destination_path` is specified, the content is written into the specified subdirectory in the resource group directory. If `force` is `False`, existing files are not replaced, but are replaced if `force` is `True`. Returns `True` if any files are copied.

copy_file(source_path, relative_destination_path, force=False) | Copies a file to the resource group directory. The `relative_destination_path` parameter can contain the destination file name or a destination path and file name. If `force` is `False`, existing files are not replaced but are replaced if `force` is `True`. Returns `True` if the file is copied.

create_file(relative_destination_path, initial_content, force=False) | Creates a file in the resource group directory. The `relative_destination_path` parameter can contain the destination file name or a destination path and file name. The `initial_content` parameter should contain the initial text content for the file. If `force` is `False`, existing files are not replaced but are replaced if `force` is `True`. Returns `True` if the file is created.

directory_path | The full path to the resource group directory.

game_cpp_code_path | The full path to the directory where the C++ code associated with the resource group should be written. This parameter is used when service API client code is generated.

get_pending_resource_status(deployment_name) | Gets a dictionary that describes the resource group's pending resource status. The status is determined by comparing the configured resource definitions, Lambda code, and parameter values to the current definitions, code, and values.

get_stack_id(deployment_name, optional=False) | Gets the resource group's stack ID for the specified deployment. If no stack exists for the resource group, raises a `HandledError` or returns `None` if optional is `True`. 

Version 1.28
2278
### get_stack_parameters(deployment_name, uploader)
Get stack parameter for the specified deployment. The uploader parameter must be an Uploader object, which is used to determine the values for the ConfigurationBucket and ConfigurationKey parameters.

### get_template_with_parameters(deployment)
Returns a Python object that contains the contents of the resource group's resource-template.json file. The template's default parameter values are set according to the parameter configuration for the specified deployment.

### is_gem
True if the resource group is defined by a gem; False if the resource group is defined in the project's resource-group directory.

### name
The name of the resource group.

### remove_output(logical_id)
Removes an output value from the resource group's resource template. Returns True if the output value existed and was removed. To save the changes, call save_template.

### remove_parameters(parameter_names)
Removes parameter definitions from the resource group's resource template. The parameter_names parameter must be a list of the names of the parameters to remove. Returns True if the parameter definitions are removed.

### remove_resources(resource_names)
Removes resource definitions from the resource group's resource template. The resource_names parameter must be a list of the logical names of the resources to remove. Returns True if the resource definitions are removed. To save the changes, call save_template.

### save_template()
Saves the current value of the template property to the resource group's resource-template.json file.

### template
A Python object that contains the content of the resource group's resource-template.json file.

### template_path
The full path to the resource group's resource-template.json file.

Other properties or functions of this object are internal to resource manager and should not be used.

**ResourceGroupContext Object**

The ResourceGroupContext object provides access to resource group configuration data. To get a ResourceGroupContext object, use the resource_groups property of the Context object.
Name | Description
--- | ---
get(name, optional=False) | Returns a ResourceGroup object for the specified resource group. If the resource group doesn't exist, raises a HandledError or none if optional is True.
keys() | Returns the names of the resource groups in a list.
values() | Returns a list of ResourceGroup objects.

Other properties or functions of this object are internal to resource manager and should not be used.

**StackContext Object**

The StackContext object provides a number of helper functions that are useful when working with AWS CloudFormation stacks. To get a StackContext object instance, use the stack property of the Context object.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| confirm_stack_operation(stack_id, stack_description, args, pending_resource_status, ignore_resource_types = []) | Displays pending changes and a confirmation prompt for the pending stack operation. 

The args parameter should be the parsed command line arguments for the command. It is used to avoid displaying prompts if the necessary --confirm-... options are provided.

The ignore_resource_types parameter is a list of resource types that are not included in the list of changes shown to the user. This parameter filters changes to the Custom::EmptyDeployment resource.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| create_using_template(stack_name, template_body, region, created_callback=None, capabilities=[]) | Uses the provided template to start a stack creation operation. Displays stack events until the create operation is complete. If the value for created_callback is not None, the value must be a called function that has the created stack ID as its only parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| create_using_url(stack_name, template_url, region, parameters=None, created_callback=None, capabilities=[]) | Uses a template identified by a URL to start a stack creation operation. Displays stack events until the create operation is complete. If the value for created_callback is not None, the value must be a called function that has the created stack ID as its only parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| delete(stack_id, pending_resource_status=None) | Starts a stack delete operation and displays stack events until the delete operation is complete.

If provided, the pending_resource_status parameter specifies the resources to be deleted from an Amazon S3 bucket in preparation for deletion of the bucket itself. To retrieve a value for this parameter, use the
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_pending_resource_status function on a ResourceGroup object, or</td>
<td>get_pending_resource_status function on the StackContext object. Preparing an S3 bucket for deletion deletes all the objects from the bucket unless the bucket specifies a DeletionPolicy property value of Retain.</td>
</tr>
<tr>
<td>use the get_pending_resource_status function on the StackContext</td>
<td></td>
</tr>
<tr>
<td>object. Preparing an S3 bucket for deletion deletes all the objects</td>
<td></td>
</tr>
<tr>
<td>from the bucket unless the bucket specifies a DeletionPolicy</td>
<td></td>
</tr>
<tr>
<td>property value of Retain.</td>
<td></td>
</tr>
<tr>
<td>describe_resources(stack_id, recursive=True, optional=False)</td>
<td>Uses the AWS CloudFormation describe_stack_resources API to return a description of a stack's resources. The content of the StackResources structure is converted to a dictionary keyed on the resource's logical ID.</td>
</tr>
<tr>
<td></td>
<td>If recursive is True, the descriptions of the resources in the nested stacks are also returned. In this case the resource's logical ID has the form <code>&lt;nested-stack-logical-id&gt;.&lt;nested-resource-logical-id&gt;</code>.</td>
</tr>
<tr>
<td></td>
<td>If the stack does not exist, a ValueError is raised if optional is False. Otherwise, an empty dictionary is returned.</td>
</tr>
<tr>
<td>describe_stack(stack_id, optional=False)</td>
<td>Returns a dictionary that contains the data returned by a call to the AWS CloudFormation describe_stacks API. If the stack does not exist, raises a ValueError or returns None if optional is True.</td>
</tr>
<tr>
<td>get_current_parameters(stack_id)</td>
<td>Gets a stack's current AWS CloudFormation template parameter values.</td>
</tr>
<tr>
<td>get_current_template(stack_id)</td>
<td>Gets a Python object that contains the stack's current AWS CloudFormation template.</td>
</tr>
<tr>
<td>get_pending_resource_status(stack_id, new_template={}, new_parameter_values={}, new_content_paths={})</td>
<td>Gets a dictionary that describes a stack's pending resource status. The status is determined by comparing the configured resource definitions, Lambda code, and parameter values to the current definitions, code, and values.</td>
</tr>
<tr>
<td></td>
<td>The new_template parameter is the resource definition template to which the stack's current template will be compared.</td>
</tr>
<tr>
<td></td>
<td>The new_parameter_values parameter is a dictionary of parameter values that will be compared to the stack's current parameter values.</td>
</tr>
<tr>
<td></td>
<td>The new_content_paths parameter is a dictionary that maps logical resource names to a list of full directory or file paths. The paths specify where content related to each resource is stored. A resource has pending update status if any of these files have changed since the last resource status update.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>get_physical_resource_id(stack_id,</td>
<td>Gets the physical ID of a resource in the specified stack. If expected_type is specified, the resource type is verified to be that type. If the type returned is not the type specified, raises a HandledError. If the specified resource does not exist, raises a HandledError or returns None if optional is True.</td>
</tr>
<tr>
<td>logical_resource_id, expected_type=None,</td>
<td></td>
</tr>
<tr>
<td>optional=False)</td>
<td></td>
</tr>
<tr>
<td>get_resource_arn(stack_id, logical_resource_id)</td>
<td>Gets the ARN for a resource in the specified stack.</td>
</tr>
<tr>
<td>get_stack_status(stack_id)</td>
<td>Gets status information for the stack. This function calls the AWS CloudFormation describe_stacks API and returns the StackStatus property of the first entry in the Stacks array of the response.</td>
</tr>
<tr>
<td>id_exists(stack_id)</td>
<td>Determines whether a stack with the specified ID (ARN) exists.</td>
</tr>
<tr>
<td>name_exists(stack_name, region)</td>
<td>Determines whether a stack with the specified name exists in the specified region.</td>
</tr>
<tr>
<td>update(stack_id, template_url, parameters={}, pending_resource_status={}, capabilities={})</td>
<td>Starts a stack update operation and displays stack events until the update operation is complete. If provided, the pending_resource_status parameter specifies the resources to be deleted from an S3 bucket in preparation for deletion of the bucket itself. To retrieve a value for this parameter, use the get_pending_resource_status function on a ResourceGroup object. Or use the get_pending_resource_status function on the StackContext object. Preparing an S3 bucket for deletion deletes all the objects from the bucket unless the bucket specifies a DeletionPolicy property value of Retain.</td>
</tr>
</tbody>
</table>

Other properties or functions of this object are internal to resource manager and should not be used.

**ViewContext Object**

The ViewContext object contains methods that produce Cloud Canvas Resource Manager output messages. Hook functions typically do not require these functions. For more information, see the source code for this object.

**Uploader Object**

You can use an Uploader object function to upload content to the project global area of the project's Configuration bucket. You pass Uploader object instances to the hook functions before_project_update, after_project_update, before_resource_group_update, and after_resource_group_update.
### Context

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>The current Context object.</td>
</tr>
<tr>
<td>bucket</td>
<td>The name of the project's Configuration bucket.</td>
</tr>
<tr>
<td>key</td>
<td>The object name prefix that is used when naming uploaded objects.</td>
</tr>
</tbody>
</table>

#### upload_content(name, content, description)

Uploads the specified content using `key` + '/ + name as the object name. A description of the upload is shown to the user.

#### upload_file(name, path)

Uploads a file using `key` + '/ + name as the object name.

#### upload_dir(name, path, alternate_root = None)

Uses `key` + '/' + name as the base object name to recursively upload the contents of a directory. If `alternate_root` is not None, the value specified is used as the object name prefix instead of `key`.

#### zip_and_upload_directory(directory_path, file_name=None, aggregated_directories=None, aggregated_content=None)

Recursively compresses the contents of a directory into a .zip file. It uses `key` + '/' + directory-name.zip as the object name to upload the file. The directory-name is the name of the directory at the end of directory_path.

You can use the file_name argument to override the directory-name.zip part of the object name.

The aggregated_directories argument can be an optional dictionary that specifies the paths of additional directories whose contents are included in the .zip file. The keys are the path location in the .zip file where the content is put.

The aggregated_content argument can be a dictionary that contains additional content to include in the .zip file. The keys are the path location in the .zip file where the content is put.

### Update Hook Functions

Update hooks are implemented in a cloud gem's Gem\<gem-name>\AWS\resource-manager-code\update.py file. If update hooks are defined in the module, the resource manager uses the parameters that are described in the Hook Function Parameters (p. 2266) section to call the following functions.

#### after_project_updated

The after_project_updated hook function is called after a project stack update operation finishes successfully.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project_uploader</td>
<td>An Uploader object that you can use to upload additional data.</td>
</tr>
</tbody>
</table>
after_resource_group_updated

The after_resource_group_updated hook function is called after a resource group stack update operation finishes successfully.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment_name</td>
<td>The name of the deployment that was updated.</td>
</tr>
<tr>
<td>resource_group_name</td>
<td>The name of the resource group that was updated.</td>
</tr>
<tr>
<td>resource_group_uploader</td>
<td>An Uploader object that you can use to upload additional data.</td>
</tr>
</tbody>
</table>

before_project_updated

The before_project_updated hook function is called before a project stack update operation begins.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project_uploader</td>
<td>An Uploader object that you can use to upload data for the operation.</td>
</tr>
</tbody>
</table>

before_resource_group_updated

The before_resource_group_updated hook function is called before a resource group stack update operation begins.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment_name</td>
<td>The name of the deployment being updated.</td>
</tr>
<tr>
<td>resource_group_name</td>
<td>The name of the resource group being updated.</td>
</tr>
<tr>
<td>resource_group_uploader</td>
<td>An Uploader object that can be used to upload additional data.</td>
</tr>
</tbody>
</table>

gather_writable_check_list

The gather_writable_check_list hook function is called before an update operation to gather a list of writable files. If any of the local files to be updated are read-only, the resource manager gives the user an opportunity to make the files writable (for example, with a source control system).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_list</td>
<td>A list of the full paths of writeable files.</td>
</tr>
</tbody>
</table>

Command Hook Functions

If the following command line hook functions exist in a module, they are defined in a cloud gem's Gem \
<gem-name>\AWS\resource-manager-code\command.py file. The resource manager uses the parameters in the Hook Function Parameters (p. 2266) section to call the following functions.
add_cli_commands

Adds additional commands to the command line parser. Called before command line argument parsing. Cloud Canvas Resource Manager uses the Python `argparse` module for command line parsing. All commands are grouped into a number of different subparsers. For example, the commands `lmbr_aws project list-resources` and `lmbr_aws deployment list` contain the subparsers `project` and `deployment`, which define a `list-resources` and `list` command, respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subparsers</td>
<td>The subparser collection object returned by the <code>argparse add_subparsers</code> function. Use the hook function to add a subparser to the collection. Then, add the commands to the subparser.</td>
</tr>
<tr>
<td>add_common_args</td>
<td>Adds a set of common arguments to a command. This function adds the following arguments. For a description of these arguments, see Using the Cloud Canvas Command Line (p. 2390).</td>
</tr>
</tbody>
</table>

  --aws-access-key  
  --aws-secret-key  
  --profile  
  --assume-role  
  --root-directory  
  --game-directory  
  --aws-directory  
  --user-directory  
  --verbose  
  --no-prompt

The common arguments are processed by `lmbr_aws`. The hook does not typically process these options.

This function can take the following parameters:

- `parser` – The `argparse` defined parser object to which the arguments are added.
- `no_assume_role` – True specifies that the `--assume-role` common argument is not added; False specifies that it is.

add_cli_view_commands

Adds additional methods to the `ViewContext` object. Called before other command line commands.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_context</td>
<td>A ViewContext object.</td>
</tr>
</tbody>
</table>

**add_gui_commands**

Adds commands to the Cloud Canvas Resource Manager in Lumberyard Editor. When the resource manager window is first opened, Lumberyard Editor initializes the resource manager Python subsystem, which calls `add_gui_commands`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handlers</td>
<td>A dictionary that maps user interface command names to the handler functions that process them. The command names are passed to Python from the user interface.</td>
</tr>
</tbody>
</table>

**add_gui_view_commands**

Adds additional methods to the `ViewContext` object. Called before a GUI command is executed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_context</td>
<td>A ViewContext object.</td>
</tr>
</tbody>
</table>

### Running AWS API Jobs Using the Cloud Gem Framework

The Cloud Gem Framework and this documentation are in preview release and are subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

The Cloud Gem Framework Gem provides C++ classes that can execute any C++ AWS API call by using the Lumberyard job execution system. This allows the operation to be performed on background threads that are managed by the job system.

**To use AWS API Jobs in your project**

1. In the [Creating Lumberyard projects](#) (p. 43), enable the Cloud Canvas Common and Cloud Gem Framework gems for your project.
2. We recommend that you put the code that uses AWS in a gem, but this is not required. If you do use a gem, make the Cloud Gem Framework and Cloud Canvas Common gems dependencies of your gem by adding the following to your gem's `gem.json` file.

```json
"Dependencies": [
  {
    "Uuid" : "6fc787a982184217a5a553ca24676cfa",
```
3. Activate your gem for your project.

4. In your gem or game project's `.wscript` file, make the following changes:

   a. To the list of includes, add:

   ```
   bld.Path('Code/SDKs/AWSNativeSDK/include')
   ```

   b. To the list of used static libraries, add `CloudGemFrameworkStaticLibrary`.

   c. Add `AWS_CPP_SDK_CORE` and other AWS API dynamic libraries as required. For a list of available aliases like `AWS_CPP_SDK_LAMBDA` and other library names, see the `dev\_WAF\_3rd_party\aws_native_sdk_shared.json` file.

   d. Add the security libraries for operating systems other than Windows, as in the following `.wscript` file for a gem.

   ```python
   SUBFOLDERS = []
   def build(bld):
       import lumberyard_sdks
       bld.DefineGem(  
           includes = [bld.Path('Code/SDKs/AWSNativeSDK/include')],  
           file_list = ['cloudcanvassample.waf_files'],  
           use = ['CloudGemFrameworkStaticLibrary'],  
           uselib = ['AWS_CPP_SDK_CORE', 'AWS_CPP_SDK_LAMBDA'],  
           darwin_lib = ['curl'],  
           linux_lib = ['curl'],  
           ios_lib = ['curl'],  
           appletv_lib = ['curl'],  
           ios_framework = ['security'],  
           appletv_framework = ['security']  
       )
       bld.recurse(SUBFOLDERS)
   ```

5. Include the `CloudGemFramework\AwsApiJob.h` header and the AWS SDK header files that are required for calling an API, as in the following example:

   ```
   #include <CloudGemFramework/AwsApiRequestJob.h>

   #pragma warning(disable: 4355) // <future> includes ppltasks.h which throws a C4355 warning: 'this' used in base member initializer list
   #include <aws/lambda/LambdaClient.h>
   #include <aws/lambda/model/InvokeRequest.h>
   #include <aws/lambda/model/InvokeResult.h>
   #include <aws/core/utils/Outcome.h>
   #include <aws/core/utils/memory/stl/AWSStringStream.h>
   #pragma warning(default: 4355)
   ```
6. Using code similar to the following, run an AWS API job. An alternative approach is to extend the job class (like `LambdaInvokeRequestJob` in the example) and provide overrides for the `OnSuccess` and `OnFailure` methods.

```cpp
using LambdaInvokeRequestJob = AWS_API_REQUEST_JOB(Lambda, Invoke);

auto job = LambdaInvokeRequestJob::Create(
    [](LambdaInvokeRequestJob* job) // OnSuccess handler - runs on job thread
    {
        Aws::IOStream& stream = job->result.GetPayload();
        std::istreambuf_iterator<AZStd::string::value_type> eos;
        AZStd::string content =
            AZStd::string{std::istreambuf_iterator<AZStd::string::value_type>(stream), eos};
        AZ_Printf("Example", "Got response %s", content.c_str());
    },
    [](LambdaInvokeRequestJob* job) // OnError handler (optional) - runs on job thread
    {
        AZ_Printf("Example", "Was error %s", job->error.GetMessageA().c_str());
    });

AZStd::string content = "...";

std::shared_ptr<Aws::StringStream> stream = std::make_shared<Aws::StringStream>();
*stream << content.c_str();

job->request.SetFunctionName("...");
job->request.SetBody(stream);
job->Start();
```

7. If your project uses the **Cloud Canvas Resource Manager**, get the physical resource ID and the logical resource ID of the AWS resource for each resource group. These IDs cause your AWS API call to use the correct resource for the active deployment. This ensures that your development, test, and released versions of a game don’t interfere with each other.

```cpp
#include <CloudCanvasCommon/CloudCanvasCommonBus.h>

AZStd::string functionName;
EBUS_EVENT_RESULT(functionName, CloudCanvasCommon::CloudCanvasCommonRequestBus,
    GetLogicalToPhysicalResourceMapping, "RESOURCE-GROUP.RESOURCE");

job->request.SetFunctionName(functionName.c_str());
```

8. If your project uses the **Cloud Canvas Resource Manager**, the AWS API is called using the player’s AWS credentials. These credentials are provided by the anonymous Amazon Cognito Identitypool that Cloud Canvas creates for your project. If you do not use **Cloud Canvas Resource Manager** or want to use other credentials, you can use code like the following to override the default configuration.

```cpp
#include <aws/core/auth/AWSCredentialsProvider.h>

LambdaInvokeRequestJob::Config config = LambdaInvokeRequestJob::GetDefaultConfig();
const char* accessKey = "...";
const char* secretKey = "...";

config.credentialsProvider = std::make_shared<Aws::Auth::SimpleAWSCredentialsProvider>(accessKey, secretKey);
config.requestTimeoutMs = 20000;

auto job = LambdaInvokeRequestJob::Create(
    ..., // OnSuccess handler
    ..., // OnError handler
    &config
);
AWS Behavior Context Reflections

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The CloudGemAWSScriptBehaviors gem uses the behavior context to expose AWS services such as Amazon Cognito, AWS Lambda, Amazon S3, and HTTP utilities to script. The `dev\CloudGemSamples\Scripts` directory has sample Lua code for each reflection. To try the samples, run the AWSBehaviorExamples level in the CloudGemSamples project.

**API Service**

API service behavior context reflections include a class and EBus that interact with Amazon API Gateway.

**AWSBehaviorAPI Class**

Calls methods that are exposed through the Amazon API Gateway service. The class properly signs requests with appropriate credentials in an HTTP request. The class translates the logical resource names in Cloud Canvas to physical names. This makes the class easy to use with other cloud gems.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string HTTPMethod</td>
<td>Specifies the HTTP method that calls the API. This string is API specific. Valid strings are: GET, POST, DELETE, PUT, and PATCH.</td>
</tr>
<tr>
<td>string Query</td>
<td>A standard URL query string that passes parameters to the API. This string is only the query and function portion of the URL and has the format <code>player/messages?time=now&amp;lang=eng</code>.</td>
</tr>
<tr>
<td>string ResourceName</td>
<td>Specifies the logical resource name of the API to call. Accepts only API Gateway resource names. Returns an error if other resource types are specified.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Execute()</td>
<td>Performs the HTTP request with the specified properties.</td>
</tr>
</tbody>
</table>

**AWSBehaviorAPINotificationsBus**

The `AWSBehaviorAPINotificationsBus` has the following methods.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void GetResponse(int responseCode, string responseData)</td>
<td>Called when the HTTP request returns a response. The <code>responseCode</code> parameter contains the HTTP response status code.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>response code; the <code>responseData</code> parameter contains the response data, which is often a JSON string.</td>
</tr>
<tr>
<td>void OnError(string <code>requestBody</code>)</td>
<td>Called when the API call fails. The <code>requestBody</code> parameter contains the request body.</td>
</tr>
<tr>
<td>void OnSuccess(string <code>resultBody</code>)</td>
<td>Called when the API call succeeds. The <code>resultBody</code> parameter contains the request response body.</td>
</tr>
</tbody>
</table>

**Lambda Service**

Lambda service behavior context reflections include a class and EBus that interact with AWS Lambda.

**AWSLambda Class**

Invokes an AWS Lambda function.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string <code>functionName</code></td>
<td>The logical function name in Cloud Canvas that you want to invoke.</td>
</tr>
<tr>
<td>string <code>requestBody</code></td>
<td>Optional. Specifies request parameters to pass to the Lambda invocation.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void InvokeAWSLambda()</td>
<td>Invokes the specified Lambda function.</td>
</tr>
</tbody>
</table>

**AWSLambdaHandler**

The `AWSLambdaHandler` EBus has the following methods.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void OnError(string <code>errorBody</code>)</td>
<td>Called when the Lambda invocation fails. The <code>errorBody</code> parameter contains the information that was returned in the Lambda error response.</td>
</tr>
<tr>
<td>void OnSuccess(string <code>resultBody</code>)</td>
<td>Called when the Lambda invocation succeeds. The <code>resultBody</code> parameter contains the information that was returned by the Lambda function, if any.</td>
</tr>
</tbody>
</table>

**S3 Service**

S3 behavior context reflections include classes and EBuses that interact with Amazon S3.
AWSBehaviorS3Upload Class

 Uploads a local file to an Amazon S3 bucket.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string bucketName</td>
<td>Specifies the logical name in Cloud Canvas for the destination Amazon S3 bucket.</td>
</tr>
<tr>
<td>string contentType</td>
<td>Specifies the MIME file type of the file to be uploaded.</td>
</tr>
<tr>
<td>string keyName</td>
<td>Specifies the destination key for the file. The destination key is the file name that appears in Amazon S3 and must be unique to the bucket.</td>
</tr>
<tr>
<td>string localFileName</td>
<td>Specifies the fully qualified path name of the local file to be uploaded.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Upload()</td>
<td>Uploads the file with the specified properties.</td>
</tr>
</tbody>
</table>

AWSBehaviorS3UploadNotificationsBus

The AWSBehaviorS3UploadNotificationsBus has the following methods.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void OnError(string errorBody)</td>
<td>Called when the file upload fails. The errorBody parameter contains the error information returned by Amazon S3, if any.</td>
</tr>
<tr>
<td>void OnSuccess(string resultBody)</td>
<td>Called when the file upload succeeds. The resultBody parameter contains the information that is returned by Amazon S3, if any.</td>
</tr>
</tbody>
</table>

AWSBehaviorS3Download Class

Downloads a file from an Amazon S3 bucket to the local file system.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string bucketName</td>
<td>Specifies the logical name in Cloud Canvas for the Amazon S3 source bucket.</td>
</tr>
<tr>
<td>string keyName</td>
<td>Specifies the key name (the file name) of the Amazon S3 source file.</td>
</tr>
<tr>
<td>string localFileName</td>
<td>Specifies the fully qualified path of the location to which the downloaded file is written.</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Download()</td>
<td>Downloads the file with the specified properties.</td>
</tr>
</tbody>
</table>

#### AWSBehaviorS3DownloadNotificationsBus

The **AWSBehaviorS3DownloadNotificationsBus** has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void OnError(string errorBody)</td>
<td>Called when the file download fails. The <code>errorBody</code> parameter contains error information returned by Amazon S3.</td>
</tr>
<tr>
<td>void OnSuccess(string resultBody)</td>
<td>Called when the file download succeeds. The <code>resultBody</code> parameter contains information returned by Amazon S3.</td>
</tr>
</tbody>
</table>

#### AWSBehaviorS3Presign Class

Provides a presigned URL for a specified Amazon S3 file. This is a URL that can be shared that allows authenticated access to the specified file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string bucketName</td>
<td>Specifies the logical name in Cloud Canvas for the target S3 bucket.</td>
</tr>
<tr>
<td>string keyName</td>
<td>Specifies the key name (the file name) of the target Amazon S3 file.</td>
</tr>
<tr>
<td>string requestMethod</td>
<td>The HTTP method to generate for the URL. Valid strings are PUT, POST, DELETE, and GET.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Presign()</td>
<td>Gets a presigned URL for the specified Amazon S3 file.</td>
</tr>
</tbody>
</table>

#### AWSBehaviorS3PresignNotificationsBus

The **AWSBehaviorS3PresignNotificationsBus** has the following methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void OnError(string errorBody)</td>
<td>Called when the operation fails. The <code>errorBody</code> parameter contains error information returned by Amazon S3.</td>
</tr>
</tbody>
</table>
Utilities

Utility classes and EBuses provide functionality for HTTP, JSON, URL, and string operations.

AWSBehaviorHTTP Class

The AWSBehaviorHTTP class implements a simple general-purpose HTTP GET request. You can use the class to get any webpage or presigned URL.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string URL</td>
<td>The URL to get.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Get()</td>
<td>Gets the URL specified by the URL property.</td>
</tr>
</tbody>
</table>

AWSBehaviorHTTPNotificationsBus

The AWSBehaviorHTTPNotificationsBus has the following methods.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void OnSuccess(string resultBody)</td>
<td>Called when the GET operation succeeds. The resultBody parameter contains the request response body.</td>
</tr>
<tr>
<td>void OnError(string errorBody)</td>
<td>Called when the GET operation fails. The errorBody parameter contains the error body.</td>
</tr>
</tbody>
</table>
| void GetResponse(int responseCode, StringMap headerMap, string contentType, string responseBody) | Called when the request returns a response. The parameters contain the following information:  
  * headerMap – Contains all the header information. It can be passed to a StringMap (p. 2295) map property.  
  * contentType – Contains the MIME content type for the response data.  
  * responseBody – Contains the response data in string format. |
**JSON Class**

Traverses and parses JSON data. This class reads but does not write JSON data. To write JSON data, use the string utilities provided in the language that you are using.

Some languages do not provide iterator support for JSON. To accommodate these languages, the JSON class maintains a current value state that gets updated as it traverses the tree of JSON data. Because JSON data consists of objects and arrays, the current value is either the value of the current object key or the value at the current position in the array. For examples, see the sample code in the \dev\CloudGemSamples\Scripts\AWSBehaviorJSONTest.lua file.

The JSON class has no properties. The following table lists its methods.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int EnterArray()</td>
<td>If the current value is an array, sets the current value to the first element of the array and returns the number of array elements.</td>
</tr>
<tr>
<td>void EnterObject(string key)</td>
<td>If the current value is an object, sets the current value to the value of the specified key.</td>
</tr>
<tr>
<td>void ExitArray()</td>
<td>If the current value was set by using EnterArray, sets the current value to the array that was entered before the call to EnterArray.</td>
</tr>
<tr>
<td>void ExitCurrentObject()</td>
<td>If the current value was set by using EnterObject, sets the current value to the object that was entered before the call to EnterObject.</td>
</tr>
<tr>
<td>void FromString(string JSONString)</td>
<td>Loads the JSON object from a JSON string. Parsing errors are logged to the console.</td>
</tr>
<tr>
<td>boolean GetBoolean()</td>
<td>If the current value is a Boolean, returns the value of the Boolean. If the current value is not a Boolean, returns false.</td>
</tr>
<tr>
<td>double GetDouble()</td>
<td>If the current value is a double, returns the value of the double. If the current value is not a double, returns 0.</td>
</tr>
<tr>
<td>integer GetInteger()</td>
<td>If the current value is an integer, returns the value of the integer. If the current value is not an integer, returns 0.</td>
</tr>
<tr>
<td>string GetString()</td>
<td>If the current value is a string, returns the value of the string. If the current value is not a string, returns an empty string.</td>
</tr>
<tr>
<td>boolean IsArray()</td>
<td>Returns true only if the current value is an array.</td>
</tr>
<tr>
<td>boolean IsBoolean()</td>
<td>Returns true only if the current value is a Boolean.</td>
</tr>
<tr>
<td>boolean IsDouble()</td>
<td>Returns true only if the current value is a double.</td>
</tr>
<tr>
<td>boolean IsInteger()</td>
<td>Returns true only if the current value is an integer.</td>
</tr>
<tr>
<td>boolean IsObject()</td>
<td>Returns true only if the current value is an object.</td>
</tr>
<tr>
<td>boolean IsString()</td>
<td>Returns true only if the current value is a string.</td>
</tr>
<tr>
<td>void LogToDebugger()</td>
<td>Displays the entire JSON object in the console in human readable format.</td>
</tr>
<tr>
<td>boolean NextArrayItem()</td>
<td>Sets the current value to the next element of the array. If the current element is the last element in the array, returns false.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>string ToString()</td>
<td>Returns the entire JSON object as a string.</td>
</tr>
</tbody>
</table>

The JSON class has no corresponding EBus.

**AWSBehaviorURL Class**

The `AWSBehaviorURL` class provides "URL decode" functionality. It removes any escape characters from the specified URL and returns the resulting string.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string URL</td>
<td>The URL to decode.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Decode()</td>
<td>Decodes the specified URL.</td>
</tr>
</tbody>
</table>

**AWSBehaviorURLNotificationsBus**

The `AWSBehaviorURLNotificationsBus` has the following methods.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void OnError(string error)</td>
<td>Called if the URL was not successfully decoded. The <code>error</code> parameter contains the reasons for the failure.</td>
</tr>
<tr>
<td>void OnSuccess(string result)</td>
<td>Called if the URL was successfully decoded. The <code>result</code> parameter contains the decoded string.</td>
</tr>
</tbody>
</table>

**StringMap Class**

A simple mapping of string key–value pairs. The `StringMap` class is most often used with the HTTP Get class but is useful for general string map operations.

**Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>Contains a string map that was received from another method for parsing.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Clear()</td>
<td>Removes all key–value pairs from the map.</td>
</tr>
</tbody>
</table>
### StringMap Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int GetSize()</code></td>
<td>Returns the number of key–value pairs currently stored in the map.</td>
</tr>
<tr>
<td><code>string GetValue(string key)</code></td>
<td>Returns the value at the specified key. If the key that was specified doesn't exist, returns an empty string without adding the key to the map.</td>
</tr>
<tr>
<td><code>boolean HasKey(string key)</code></td>
<td>Returns true if the map contains the specified key.</td>
</tr>
<tr>
<td><code>void LogToDebugger()</code></td>
<td>Displays all key–value pairs in the console.</td>
</tr>
<tr>
<td><code>void RemoveKey(string key)</code></td>
<td>Removes the key–value pair from the map that corresponds to the specified key. If the specified key does not exist, fails without returning an error.</td>
</tr>
<tr>
<td><code>void SetValue(string key, string value)</code></td>
<td>Sets the value of the specified key to the specified value. If the key doesn't exist, the key–value pair is added to the map. If the key does exist, its existing value is overwritten with the specified value.</td>
</tr>
</tbody>
</table>

The `StringMap` class has no corresponding EBus handlers.

### Adding AWS Resources to a Cloud Gem

**Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.**

Cloud Canvas cloud gems can be used out of the box, without interacting with the code and the Cloud Gem Framework (p. 2257) that powers them. However, you might be interested in modifying existing cloud gems and creating your own cloud gems, possibly for distribution to others. If so, you might need to add support for additional AWS CloudFormation types beyond the ones natively supported by Cloud Canvas. This topic provides information on how to do that.

Cloud Canvas gems provide AWS CloudFormation templates that specify the AWS resources that the gem requires. AWS CloudFormation templates support AWS resource types, which are prefixed with `AWS::`. They also support `AWS::CloudFormation::CustomResource` custom resource types or any resource type prefixed with `Custom::`. The template file for a cloud gem is located at `lumberyard_version\dev\Gems \gem_name\AWS \resource_template.json`. The `resource_template.json` file is a specialized version of a AWS CloudFormation template that provides metadata specific to Cloud Canvas.

While AWS CloudFormation templates support a large catalog of AWS resource types, the templates for Cloud Canvas gems are more limited in scope. When creating a Cloud Canvas gem, you have the following options:

- Use one of the subset of types in the `AWS::` namespace that are directly supported by Cloud Canvas (no additional work required).
- Add support for `AWS::` types not already supported by Cloud Canvas.
- Add your own `Custom::` resource types that execute custom Lambda function code when instances of that type are created, modified or deleted.

Custom resources are a good way to integrate your in-house services and access AWS services that are not directly supported.
Topics

- Resource Type Specification (p. 2297)
- Adding Support for New AWS:: and Custom:: Types (p. 2301)
- Versioning of Custom Resources (p. 2303)

Resource Type Specification

Resource types are specified in a Cloud Canvas project-template.json file. For types that Cloud Canvas natively supports, the file is located at Gems\CloudGemFramework\version\AWS\project-template.json. The existing resource is named CoreResourceTypes.

**Note**

You should not interact with this file. You are not prevented from modifying this file to support new types or even change the support for existing types. However, your changes are not received or usable by others with whom you share your gem.

The template file for new types that you want to add support for in your gem is located at lumberyard_version\dev\Gems\gem_name\AWS\project_template.json.

Resources are specified in the template as follows:

```json
{
  "Resources": {
    "<resource-name>": {
      "Type": "Custom::ResourceTypes",
      "Properties": {
        "ServiceToken": {
          "Fn::GetAtt": [
            "ProjectResourceHandler",
            "Arn"
          ]
        },
        "LambdaConfiguration": <lambda-configuration>,
        "LambdaTimeout": <seconds>,
        "Definitions": {
          <definition>,
          <definition>,
          ...
        }
      }
    }
  }

<lambda-configuration> := {
  "Fn::GetAtt": {
    [ "<lambda-resource-name>", "ComposedLambdaConfiguration" ]
  }
}

<definition> := {
  "<type-name>": {
    "PermissionMetadata": {
      "RestrictActions": [ "<action>", "<action>", ... ],
      "DefaultRoleMappings": [ {
        "role-mapping",
        "role-mapping",
        "...
      }]}
    }
  }
}
"ArnFormat": "<arn-format>",
"ArnFunction": <lambda-function-spec>,
"HandlerFunction": <lambda-function-spec>,
"DisplayInfo": {
    "AWSConsoleUrls": [
        <console-url>
    ]
}

<console-url> := {
    "Label": "<label>",
    "Url": "<url>"
}

<lamba-function-spec> := {
    "Function": "<function-handler>",
    "HandlerPolicyStatement": [
        <policy>,
        <policy>,
        ...
    ],
}

<lamba-resource-definition> := {
    "Properties": {
        "ConfigurationBucket": {
            "Ref": "Configuration"
        },
        "ConfigurationKey": {
            "Ref": "ConfigurationKey"
        },
        "FunctionName": "<lambda-folder-name>",
        "Runtime": "python3.7",
        "ServiceToken": {
            "Fn::GetAtt": [
                "ProjectResourceHandler",
                "Arn"
            ]
        },
    }
    "Type": "Custom::LambdaConfiguration"
}

Values to Specify

Replace the following items in the template with your own values as required.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;resource-name&gt;</td>
<td>Specifies a custom name for the resource. The name must be unique in the template.</td>
</tr>
<tr>
<td>&lt;lambda-resource-name&gt;</td>
<td>Specifies the name of a Custom::LambdaConfiguration element that you include to provide Lambda code for your types. This entry is required if you specify either the ArnFunction or HandlerFunction entries.</td>
</tr>
<tr>
<td>&lt;lambda-timeout&gt;</td>
<td>Specifies the timeout, in seconds, to apply to the specified ArnFunction or HandlerFunction entries.</td>
</tr>
<tr>
<td>&lt;lambda-directory-name&gt;</td>
<td>Specifies the name of a subdirectory to add to your gem's \AWS \project-code\lambda-code directory. This subdirectory</td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>contains Lambda code that is required for the ArnFunction or HandlerFunction entries.</td>
<td></td>
</tr>
<tr>
<td>&lt;type-name&gt;</td>
<td>Specifies a AWS CloudFormation type in the AWS:: namespace or a custom type that is prefixed with Custom:: .</td>
</tr>
</tbody>
</table>

**Elements in the Type Definition**

This section describes the other elements in a type definition.

**PermissionMetadata**

Optional. Contains the metadata configuration information for an instance of the type that controls permissions.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;action&gt;</td>
<td>Specifies an IAM action that instances of the resource type are permitted to have in the role that is created for them. Wildcards are permitted.</td>
</tr>
<tr>
<td>&lt;role-mapping&gt;</td>
<td>Specifies a default Cloud Canvas role mapping (p. 2377) to apply to instances of this type.</td>
</tr>
</tbody>
</table>

**ArnFormat**

Required if ArnFunction is not specified. Specifies the ARN for instances of the resource in the <arn-format> format string. The following substitutions are allowed:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;region&gt;</td>
<td>Specifies the region of the AWS CloudFormation stack.</td>
</tr>
<tr>
<td>&lt;account-id&gt;</td>
<td>Specifies the AWS account ID.</td>
</tr>
<tr>
<td>&lt;resource-name&gt;</td>
<td>Specifies the physical resource ID of the resource instance.</td>
</tr>
</tbody>
</table>

The following example specifies the ARN for a Lambda function:

```
arn:aws:lambda:<region>;<account-id>;function:<resource-name>
```

**ArnFunction**

Required if ArnFormat is not specified. Specifies a module and function to retrieve the ARN for the resource.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;function-handler&gt;</td>
<td>Specifies a reference to a function in the string format: Module.Function. The module must be located in the resource_types subdirectory of your gem's ServiceLambda directory.</td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;policy&gt;</td>
<td>Specifies an IAM policy statement that specifies the permissions that the</td>
</tr>
<tr>
<td></td>
<td>Lambda function requires to execute. The HandlerPolicyStatement can be</td>
</tr>
<tr>
<td></td>
<td>omitted if the Lambda function requires no permissions.</td>
</tr>
</tbody>
</table>

ArnFunction is required by some resources that do not have an ARN that can be computed from the substitutions in ArnFormat, such as an Amazon SQS queue. An Amazon SQS queue requires a call to the GetQueueAttributes API to obtain its ARN. ArnFunction supports the same substitutions as ArnFormat.

**Note**

Be sure that you specify any required permissions for your function in the HandlerPolicyStatement element of the PermissionMetadata field. These permissions are shared with the HandlerFunction if one exists.

**HandlerFunction**

Required if the type begins with Custom:::. Specifies a module and function that runs custom code in response to the Create, Update, and Delete events of the custom resource.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;function-handler&gt;</td>
<td>Specifies a reference to a function in the string format: Module.Function. This module must be located in the resource_types subdirectory of your gem's ServiceLambda directory.</td>
</tr>
<tr>
<td>&lt;policy&gt;</td>
<td>Specifies an IAM policy statement that specifies the permissions that the</td>
</tr>
<tr>
<td></td>
<td>Lambda function requires to execute. The HandlerPolicyStatement can be</td>
</tr>
<tr>
<td></td>
<td>omitted if the Lambda function requires no permissions.</td>
</tr>
</tbody>
</table>

Custom resources (that is, entries in the Resources section of the form "Type": "Custom:::<name>") should define their ServiceToken to point to the Cloud Canvas ProjectResourceHandler. The ProjectResourceHandler dispatches the events to your module so that your custom code can run with the correct permissions.

Be sure to specify any required permissions for your function in the HandlerPolicyStatement element of the PermissionMetadata field. These permissions are shared with the ArnFunction if it exists. Custom resource handlers process the events as defined by AWS CloudFormation.

**DisplayInfo**

Optional. Contains user interface information to display. Currently, this consists exclusively of <console-url>, which has a <label> and a <url>. The <url> is relative to https://console.aws.amazon.com/ and can use the same substitutions as ArnFormat.

The following example specifies links to view resources in the AWS Management Console and in CloudWatch Logs.

```json
"AwsConsoleUrls": [ 
  { 
    "Label": "View resource in AWS console",
    "Url": 
      "/lambda/#/functions/<resource-name>"
  },
  
```
Adding Support for New AWS:: and Custom:: Types

The steps to create a Custom:: type and an AWS:: type are almost identical. The main difference is that a custom type must execute a Lambda function when an instance of the type is created, updated or deleted. A custom type must therefore supply values for LambdaConfiguration (optionally for LambdaTimeout) and for HandlerFunction. An AWS type needs to call a Lambda function only if it cannot supply its ARN from simple string substitution of the values in the ArnFormat field. Therefore, for most AWS types, you can omit the LambdaConfiguration, LambdaTimeout, and HandlerFunction fields from the template entirely.

To add an AWS:: resource type

1. If your gem does not have a project-template.json file in its AWS directory, create the file and give it the following structure:

   ```json
   {
     "Resources": {
     }
   }
   
   ```

2. Create a new resource in your project-template.json file of the type Custom::ResourceTypes. The type should have properties that correspond to the Resource Type Specification (p. 2297). You can use a single Custom::ResourceTypes entry for multiple resource definitions.

3. To add a resource definition with the AWS::-prefixed typename to the resource, do one of the following:

   a. If the ARN is easily computable from the region, account ID, and resource name fields, specify the corresponding formatting pattern in the ArnFormat field.

   b. If the ARN is not easily computable from the region, account ID, and resource name fields, perform the following steps:

      a. Create a uniquely named `lambda-directory-name` directory for your gem as follows:

         ```bash
         AWS\project-code\lambda-code\lambda-directory-name\resource_types
         ```

      b. In your Resources block, add a resource with a custom `<lambda-resource-name>` name as the key. Create the `<lambda-resource-definition>` contents following the Resource Type Specification (p. 2297).

      c. Replace the `lambda-directory-name` in the definition with the `lambda-directory-name` that you created.

      d. Add the LambdaConfiguration field (and optionally the LambdaTimeout field) to the main Properties section of your resource definition. Replace `<lambda-resource-name>` with your `<lambda-resource-name>` in the `<lambda-configuration>` block.

      e. Write a Lambda function that returns the ARN for a resource of your type. Place the Lambda function in your gem's AWS\project-code\lambda-code\lambda-directory-name\resource_types directory.

      f. Add an ArnFunction field to your resource definition as a dictionary. In the dictionary, add the following items:

         a. In the Function field, provide the name of the handler in the format Module.Function.
• If your function requires access to any AWS services, create a PolicyStatement field and add the necessary policy statements to it.

4. Provide any relevant display information or links in the DisplayInfo field.

5. Add a resource of your AWS::-prefixed type to your resource-template.json file.

To add a Custom:: resource type

1. If your gem does not already have a project-template.json file in its AWS directory, create the file and give it the following structure:

```json
{
   "Resources": {
   }
}
```

2. Create a new resource in your project-template.json file of the type Custom::ResourceTypes. The type should have properties that correspond to the Resource Type Specification (p. 2297). You can use a single Custom::ResourceTypes entry for multiple resource definitions.

3. Create a uniquely named lambda-directory-name directory for your gem as follows:

AWS\project-code\lambda-code\lambda-directory-name\resource_types

4. In your Resources block, add a resource with a custom <lambda-resource-name> name as the key. Create the <lambda-resource-definition> contents following the Resource Type Specification (p. 2297).

5. Replace the lambda-directory-name in the definition with the lambda-directory-name that you created.

6. Add the LambdaConfiguration field (and optionally the LambdaTimeout field) to the main Properties section of your resource definition. Replace <lambda-resource-name> with your <lambda-resource-name> in the <lambda-configuration> block.

7. To add a resource definition with your own the Custom:: prefixed type name to the resource, perform the following steps:

   a. Specify any role metadata or restrictions for the type in the PermissionMetadata field.
   
   b. Provide a reference to your service API in the ServiceApi field.

   c. Write a Lambda function that processes Create, Update and Delete events for the custom resource. Place the code in your AWS\project-code\lambda-code\lambda-directory-name\resource_types directory.

   d. Add a HandlerFunction field to your resource definition as a dictionary. In the dictionary, add the following items:

      • In the Function field, provide the name of the handler in the format Module.Function.
      • If your function requires access to any AWS services, create a PolicyStatement field and add the necessary policy statements to it.

   e. If the ARN is easily computable from the region, account ID and resource name fields, specify the formatting pattern as the ArnFormat field.

   f. If the ARN is not easily computable from the region, account ID, and resource name fields, perform the following steps:

      i. Write a Lambda function that returns the ARN for a resource of your type. Place the Lambda function in your gem's AWS\project-code\lambda-code\lambda-directory-name\resource_types directory. You can use the same file as you did for your main handler and just give the function a different name (for example, arn_handler.)
ii. Add an ArnFunction field to your resource definition as a dictionary. In the dictionary, add the following items:

- In the Function field, provide the name of the handler in the format Module.Function.
- If your function requires access to any AWS services, create a PolicyStatement field and add the necessary policy statements to it.

8. Provide any relevant display information or links in the DisplayInfo field.
9. Add a resource of your Custom:: prefixed type to your resource-template.json file.

Versioning of Custom Resources

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In Lumberyard 1.15 and earlier versions, the most recent version of Lambda code was used to process custom resource instance events. This caused rollback issues when existing resources were processed by newer versions of the same code.

Starting in Lumberyard version 1.16, custom resources are locked by default to the version of the code that you used to create the resources. If the code changes and an existing resource instance is updated or deleted, the same version of the code that created the resource processes the event.

When new custom resource code is deployed, it does not replace the older code. Resources that are created with the new code will be tied to the new code version.

Version Coercion

If necessary, you can override the locking of a custom resource to the version of the code with which it was created. To specify a different Lambda function version for a custom resource instance's update and delete events, specify a value for CustomResourceVersion in the Cloud Canvas metadata that you have obtained from the AWS Lambda console.

Example

The following code specifies a value of 3 for CustomResourceVersion.

```
"MyCustomResource": {
  "Metadata": {
    "CloudCanvas": {
      "CustomResourceVersion": "3"
    }
  },
  "Type": "Custom::MyCustomType",
  "Properties": {
    "ServiceToken": {
      "Ref": "ProjectResourceHandler"
    }
  }
}
```

After you make this change, the specified version of the Lambda function will process the resource the next time that the resource is updated or deleted.

Removing the CustomResourceVersion metadata entry restores the default behavior of using the version that created the resource. Therefore, if you want a custom resource to always use a newer version of a handler, you must always specify a value for CustomResourceVersion.
Warning
Coercing a custom resource to process events with code different from the code that instantiated the resource exposes the resource group to risk of rollback failure. You are responsible for determining the version of the Lambda function that you want your instance to use. You can find the version from the AWS Lambda console.

Note
You can specify $LATEST as the value for CustomResourceVersion. However, this replicates the unsafe behavior in previous versions of Lumberyard of using the most recent Lambda code version to process custom resource instance events. This practice is not recommended for environments where failed stack updates cannot be tolerated.

Retention of Lambda Functions

Prior to Lumberyard version 1.16, if you deleted a type definition and then updated your project stack, the Lambda functions associated with the type definition were also deleted. These functions might include custom resource handler Lambda or ARN handler Lambda functions.

This behavior could result in rollback errors and a potentially unrecoverable stack when you tried to update instances of the resource type. For this reason, Lambda functions associated with resource types now remain in your account indefinitely and are deleted only when the project is deleted.

Note
Lambda functions are deleted in AWS only when you run lmbr_aws project delete. If you delete your project stack locally but not in AWS, the Lambda functions associated with your resource types will remain. To delete them, you can use the AWS console or the Cloud Canvas cleanup tool (p. 2362). Because the cleanup tool can be very destructive, use it with caution, and do not use it on production deployments.

Removing Unused Custom Resource Code

As you develop new custom resources or upgrade existing ones, Lambda function versions that are no longer referenced by any instances can remain in your AWS account. In most cases, the existence of these orphaned Lambda functions should not affect your AWS account's Lambda Limits.

However, if you need to remove such instances, you can run the following command:

```
lmbr_aws project clean-custom-resources
```

This command deletes all custom resource Lambda functions in a project that satisfy the following criteria:

- The function is not the most recent version.
- No existing custom resource instances were created using that version.
- No custom resource instances are currently using that version through the version coercion (p. 2303) technique.

Physical IDs

Previously, the PhysicalResourceId that a custom resource handler returned was the same as the physical ID of the resource in its AWS CloudFormation stack.

With the advent of versioning in Lumberyard 1.16, custom resources embed their own version information in AWS CloudFormation's PhysicalResourceId. If you are writing a Lambda function and need to extract the embedded ID from the AWS CloudFormation physical ID, use the `get_embedded_physical_id` function, as in the following example.
from cgf_utils import custom_resource_utils
actual_physical_id = custom_resource_utils.get_embedded_physical_id(stack_physical_id)

To access this module, make sure the .import file in your Lambda function directory includes CloudGemFramework_Utils.

For source code, see the lumberyard_version\dev\Gems\CloudGemFramework\vN\AWS\common-code\Utils\cgf_utils\cgf_utils.custom_resource_utils.py module.

ARN Handlers

Unlike custom resource handlers, ARN handlers are not version-locked. Determining an ARN is typically a simple operation that does not require versioning.

Backwards Compatibility

Custom resources that were instantiated in Lumberyard projects prior to 1.16 will continue to use the most recent version of the custom resource Lambda code to process their update and delete events. If you want the safety benefits of versioning, you must delete existing custom resource instances and recreate them.

Cloud Gem Framework Service API

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Topics

- Cloud Gem Swagger API Descriptions (p. 2306)
- Resources (p. 2311)
- Operations (p. 2312)
- Security (p. 2315)
- Cloud Gem Framework Extension Object (p. 2318)
- Cross-Gem Communication (p. 2319)
- Game Clients (p. 2322)
- Generated Game Client Code Example (p. 2324)
- Calling a Game API (p. 2336)
- Publishing Your API (p. 2339)

Lumberyard cloud gems provide services that configure and manage a game's operation. The services are implemented by AWS Lambda function resources. The game and tools access these services through the Amazon API Gateway. The code that you provide runs in a Lambda function. API Gateway manages access to the service, provides caching for frequently used results, and supports request throttling.

The following diagram shows how the Cloud Gem Framework service API interacts with the game client, AWS Lambda, and API Gateway.
The Lumberyard game engine and tools use API operations to communicate with your cloud gem service. These APIs are described in the cloud gem’s /dev/Gems/<gem-name>/AWS/swagger.json file. This file describes the input and output data for each of the operations that can be performed by the service. The swagger.json file uses the swagger API definition format, which is an open source framework for RESTful API operations.

The following is an example swagger.json file.

```json
{
    "swagger": "2.0",
    "info": {
        "version": "1.0.0",
        "title": "$RestApiResourceName$",
        "description": "API for the $ResourceGroupName$ service ($DeploymentName$ deployment)."
    },
    "schemes": [
        "https"
    ],
    "consumes": [
        "application/json"
    ],
    "produces": [
        "application/json"
    ],
    "x-amazon-cloud-canvas-lambda-dispatch": {
        "lambda": "$ServiceLambdaArn$"
    }
}
```
paths: {
  "/service/status": {
    "x-amazon-cloud-canvas-lambda-dispatch": {
      "module": "service_status"
    },
    "get": {
      "operationId": "get_service_status",
      "description": "Returns the service's status. Useful for testing connectivity.",
      "responses": {
        "200": {
          "description": "A successful service status response.",
          "schema": {
            "$ref": "#/definitions/ServiceStatus"
          }
        }
      }
    }
  },
  "/admin/messages": {
    "post": {
      "description": "Add a message to the message table",
      "parameters": [
        {
          "name": "msg",
          "in": "body",
          "required": true,
          "schema": {
            "$ref": "#/definitions/MessageData"
          }
        }
      ],
      "responses": {
        "200": {
          "description": "Unique ID for this new message",
          "schema": {
            "$ref": "#/definitions/DetailedMessageData"
          }
        }
      }
    },
    "get": {
      "description": "Get the list of N messages starting at a given index and filter with all, active, expired or planned",
      "parameters": [
        {
          "description": "The index number of the page to fetch first.
Example: 0",
          "name": "index",
          "in": "query",
          "required": true,
          "type": "integer"
        },
        {
          "description": "The maximum number of messages to fetch. Example: 1000",
          "name": "count",
          "in": "query",
          "required": true,
          "type": "integer"
        },
        {
          "description": "Options available here are 'active', 'planned', 'expired'. Example: active",
          "name": "filter",
          "in": "query",
          "required": false,
          "type": "string"
        }
      ]
    }
  }
}
"in": "query",
  "required": true,
  "type": "string"
}
],
"responses": {
  "200": {
    "description": "The list of detailed messages ",
    "schema": {
      "$ref": "#/definitions/DetailedMessageList"
    }
  }
}
},
"/admin/messages/{msg_id}": {
  "delete": {
    "description": "Delete an existing message",
    "parameters": [
      {
        "name": "msg_id",
        "description": "The message id to edit.",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "Status of the deletion",
        "schema": {
          "format": "string"
        }
      }
    }
  },
  "put": {
    "description": "Edit an existing message",
    "parameters": [
      {
        "name": "msg_id",
        "description": "The message id to edit.",
        "in": "path",
        "required": true,
        "type": "string"
      },
      {
        "name": "msg",
        "in": "body",
        "description": "The new localized message body.",
        "required": true,
        "schema": {
          "$ref": "#/definitions/MessageData"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Status of the addition",
        "schema": {
          "format": "string"
        }
      }
    }
  }
}
]
"/player/messages": {
  "get": {
    "description": "Get the list of messages scheduled for a given time and language",
    "parameters": [
      {
        "name": "time",
        "description": "The player's local time. Example: Jul 18 2017 13:43",
        "in": "query",
        "type": "string"
      },
      {
        "name": "lang",
        "description": "The ISO 639-1 language code. Example: en",
        "in": "query",
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "The list of messages for that time and language. If no time or lang is provided it falls back to UTC Eng",
        "schema": {
          "$ref": "#/definitions/MessageList"
        }
      }
    }
  }
},
"definitions": {
  "ServiceStatus": {
    "type": "object",
    "properties": {
      "status": {
        "type": "string"
      }
    },
    "required": ["status"]
  },
  "MessageData": {
    "type": "object",
    "properties": {
      "message": {
        "description": "Required - The localized message of the day text.",
        "type": "string"
      },
      "priority": {
        "description": "The priority in which to view the message. 0 has the highest priority.",
        "type": "integer"
      },
      "startTime": {
        "description": "The date time in which to start displaying the message. Example: Jul 18 2017 00:00",
        "type": "string"
      },
      "endTime": {
        "description": "The date time in which to stop displaying the message. Example: Jul 19 2017 16:00",
        "type": "string"
      }
    }
  }
}
Cloud Canvas uses these API descriptions to generate Lumberyard engine components that execute service APIs for the game. In addition, the API descriptions are used to configure API Gateway to work with your service.
Resources

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A cloud gem's service API is implemented based on the resources that are defined in the cloud gem's resource-template.json (p. 2237) file. The following are the key resources for a cloud gem:

- **ServiceAPI** – An AWS CloudFormation custom resource provided by the Cloud Gem Framework. The handler for this resource configures API Gateway to use the cloud gem's swagger.json file.
- **ServiceLambda** – An AWS Lambda function that implements the cloud gem's functionality.
- **ServiceLambdaConfiguration** – An AWS CloudFormation custom resource provided by Cloud Canvas Resource Manager that configures (p. 2256) the ServiceLambda resource.

To add definitions for these resources to a resource-template.json file, enter the following command.

```
lmbr_aws cloud-gem-framework add-service-api-resources --resource-group <gem-name>
```

In addition to adding the resources mentioned, the add-service-api-resources command does the following:

- Adds a swagger.json file to the cloud gem's AWS directory, if one doesn't already exist.
- Adds the ServiceAPI and ServiceLambda resources to the AccessControl resource definition's DependsOn property. This insures that the AccessControl resource is processed by AWS CloudFormation after the ServiceAPI and ServiceLambda resources have been processed.

To use the service API resources that you add to a resource-template.json file, you must upload those resources to AWS. To upload them, you can use the lmbr_aws resource#group upload command, the lmbr_aws deployment upload command, or click Upload Resources in the Resource Manager feature in Lumberyard Editor.

**Custom::ServiceApi Resource**

The handler for the Custom::ServiceApi AWS CloudFormation resource is provided by the Cloud Gem Framework. This handler creates, updates, and deletes API Gateway REST API, operation, deployment, and stage resources as needed.

ServiceApi resource definitions accept the following parameters:

```json
{
  "Resources": {
    ...
    "ServiceApi": {
      "Type": "Custom::ServiceApi",
      "Properties": {
        "ServiceToken": { "Ref": "ProjectResourceHandler" },
        "ConfigurationBucket": { "Ref": "ConfigurationBucket" },
        "ConfigurationKey": { "Ref": "ConfigurationKey" },
        "CacheClusterSize": { "Ref": "ServiceApiCacheClusterSize" },
        "CacheClusterEnabled": { "Ref": "ServiceApiCacheClusterEnabled" },
        "MethodSettings": { ... },
        "SwaggerSettings": {
          "ServiceLambdaArn": { "Fn::GetAtt": [ "ServiceLambda", "Arn" ] }
        }
      }
    }
  }
```

Version 1.28
2311
ServiceToken

Identifies the Lambda function that implements the custom resource handler.

ConfigurationBucket

Identifies the bucket that contains the uploaded swagger.json file.

ConfigurationKey

Identifies the location in the bucket where the swagger.json file is uploaded.

CacheClusterSize

Provides the API Gateway cacheClusterSize value when you create or update the API Gateway stage.

CacheClusterEnabled

Provides the API Gateway cacheClusterEnabled value when you create or update the API Gateway stage.

MethodSettings

Not implemented.

SwaggerSettings

Provides values that you insert into the uploaded swagger.json file before it is passed to API Gateway. For example, you can use $ServiceLambdaArn$ in the swagger.json file to insert the value of the SwaggerSettings ServiceLambdaArn property.

The following settings are automatically defined for you:

ResourceGroupName

The name of the resource group that is defined the ServiceApi resource.

DeploymentName

The name of the deployment that the ServiceApi resource is in.

RoleArn

The ARN of the role that grants API Gateway the permission to invoke the ServiceLambda (or other permissions configured by the Cloud Canvas Resource Manager Security System (p. 2370)).

Region

The AWS region where the RESTful API resides.

RestApiResourceName

The name to use for the API Gateway REST API resource. API Gateway takes this value from the swagger infoObject title property (set to $RestApiResourceName$ in the default swagger.json file). This is the stack name of the resource group with the ServiceApi logical resource ID appended (usually -ServiceApi).

Operations

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
A cloud gem’s service API can implement multiple distinct operations. You define operations in the `swagger.json` file by adding `operationObject` instances to a `pathItemObject`. For each operation, you can define the input data that the operation requires and the output data that it produces. The input data can include path, query, and body parameters. For more information on using swagger to define APIs, see http://swagger.io/.

API Gateway routes API requests to the cloud gem’s Lambda function. When you upload the cloud gem’s resources to AWS, the cloud gem’s `swagger.json` file is processed. This processing creates the request and response mappings that API Gateway uses to call the cloud gem’s Lambda function. This configuration is controlled by the `x-amazon-cloud-canvas-lambda-dispatch` (p. 2318) extension objects in the `swagger.json` file.

Service API operations are implemented in the `ServiceLambda` AWS Lambda function resource. The code for the Lambda function comes from the cloud gem’s `lambda-function-code` directory. The Cloud Gem Framework provides a service request dispatch module. For more information, see Request Execution (p. 2314).

The following diagram illustrates both request and upload processing.

### Topics
- Default Request Mapping (p. 2313)
- Default Response Mapping (p. 2314)
- Request Execution (p. 2314)

#### Default Request Mapping

A JSON object that implements the request operation is sent to the Lambda function. The JSON object contains the module and function name and the operation's parameters, as the following skeletal syntax shows.

```json
{
    "module": "<module-name>",
    "function": "<function-name>",
    "parameters": {
```
The default module name is derived from the operation path. A path like /player/{id}/highscores results in a module name like player_highscores. Parameters in the path are skipped. The paths /player and /player/{id} both map to the module name player. In this case the handler function in that module can use the existence of an id parameter value to determine the corresponding behavior. The path / (and /{param}) are mapped to the module name root.

The default function name is the operation name like GET, POST, or PUT.

These defaults can be overridden by specifying the module and/or function properties of an x-amazon-cloud-canvas-lambda-dispatch extension object. For more information, see Cloud Gem Framework Extension Object (p. 2318).

Parameter names are taken from the parameter definitions in the swagger.json file. Path, query, and body parameter types are supported.

Default Response Mapping

The Lambda function returns the value to the client as a JSON object such as the following:

```
{
"result":<lambda-return-value>
}
```

If the Lambda function exits with an error (for example, by raising an unhandled exception), it returns the JSON object in the following format.

```
{
"errorMessage":"<error-message>",
"errorType":"<error-type>"
}
```

If the error message received from the Lambda function starts with the text Client Error:, then an HTTP 400 response is sent to the client. The errorMessage and errorType properties received from the Lambda function are forwarded to the client.

If the error message received from the Lambda function does not start with Client Error:, or no error message is received at all, an HTTP 500 response is sent to the client. As a security measure, the errorMessage returned to the client is always An internal server error has occurred, and errorType is ServiceError. This is to avoid sending the exception message to the client. Doing so could provide information that could allow an attacker to discover exploits in your service implementation.

In both the HTTP 400 and HTTP 500 cases, no other information (such as a stack trace) is sent to the client; this also is a security measure.

Request Execution

The Cloud Gem Framework has built-in support for service API Lambda functions that are implemented in Python. No built-in support is provided for Node.js or Java Lambda functions. To support these languages, implement the Lambda function handler to look for the module and function properties on the event object that the Lambda function provides. You also might need to override the default values generated for the x-amazon-cloud-canvas-lambda-dispatch (p. 2318) module and function properties during the swagger.json file processing.
A service.py module file is provided for you in the service's AWS Lambda function. This module's dispatch function uses the module and function properties of the event object to dispatch the request to the service code that you provide. These properties are set by the request mapping that configures API Gateway.

Place your service code in your resource group's lambda-function-code/api directory. The dispatcher uses importlib.import_module to load the modules from that directory. If the specified function has the @service.api decorator, the dispatcher invokes the function on that module, as in the following example.

```python
import service
@service.api
def post(request, submission, user_id):
    ...
```

The first argument passed to the function is a dispatch.Request object. The request object has the following properties.

- **event** – The event object that AWS Lambda passes to the dispatch handler. The contents of the object are determined by the request template. Additional properties added to the template through the x-amazon-cloud-canvas-lambda-dispatch object's additional-request-template-content property are also located here.

- **context** – The Python context object that AWS Lambda passes to the dispatch handler.

The request object's parameters are passed to the handler function as keyword arguments (that is, by using Python **parameters).

**Error Handling**

A ClientError class is provided in the errors.py file. This class extends RuntimeException and ensures that the error message is prefixed with Client Error:. This triggers an HTTP 400 response from API Gateway as described in Default Response Mapping (p. 2314).

If other exceptions are raised during processing, a generic Internal Service Error message is sent to the client.

**Security**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you make APIs available on the Internet, you must be concerned with security. A best practice is to limit API access to only the people who require it. However, some APIs must be called by the game client. These APIs can be called by any game player or potentially anyone on the Internet.

For all APIs, consider the following:

- Use the access control mechanism described in the next section to limit API access to only those who require it.

- Don't trust parameter values provided by the client. Verify that the parameter values match expectations before you use them. Be careful when inserting parameter values into query strings that are sent to DynamoDB or other services. For more information, see code injection.
• API Gateway automatically protects your backend systems from distributed denial-of-service (DDoS) attacks, whether attacked with counterfeit requests (Layer 7) or SYN floods (Layer 3). However, this does not protect from less frequent requests that do not trigger API Gateway's protections. These other requests might still have a significant impact on your operating costs due to excessive I/O or on game performance.

Access Control

Configuring access control for a service API involves setting three distinct sets of permissions:

1. The `execute-api` operation, enforced by API Gateway.
2. The Lambda `Invoke` function, enforced by AWS Lambda.
3. Access to the cloud gem resource, enforced by the resource's AWS service (for example, Amazon DynamoDB or Amazon S3.)

In all three cases, you use the Cloud Canvas Resource Manager Security (p. 2370) system to configure access. This involves putting Permissions metadata on the `ServiceApi` and `ServiceLambda` resource definitions, as well as on the definitions of resources accessed by the `ServiceLambda` code. This is illustrated in the following diagram:

The permissions granted by `ServiceApi` are described in detail in the next section. `ServiceLambda` gives `ServiceApi` permission to invoke the Lambda function. Other resources give `ServiceLambda` the permissions that the Lambda function requires.

Service API Permissions

You can secure service API operations on an individual basis. For example, you could give permission to submit high scores to the game, and permission to delete fraudulent high scores to someone who manages operations.

By default, only valid AWS IAM credentials can execute operations. This is done using swagger security requirement and security definitions objects that are added to each operation during upload processing (p. 2318), unless the swagger operation object already defines a security object. You can
configure API Gateway to use other forms of security, or none at all, by putting security objects in
swagger object definitions in your swagger.json file.

To grant permission to execute an operation, modify your resource-template.json file to include
metadata on the ServiceApi resource definition, as in the following example.

```json
{
  "Resources": {
    ...,
    "ServiceApi": {
      "Type": "Custom::ServiceApi",
      "Properties": { ... },
      "Metadata": {
        "CloudCanvas": {
          "Permissions": [
            {
              "AbstractRole": ["ProjectOwner", "DeploymentOwner"],
              "Action": "execute-api:*",
              "ResourceSuffix": "/**"
            },
            {
              "AbstractRole": "Player",
              "Action": "execute-api:Invoke",
              "ResourceSuffix": "//*/POST/score/*"
            },
            {
              "AbstractRole": "DevOps",
              "Action": "execute-api:Invoke",
              "ResourceSuffix": "//*/DELETE/score/*"
            }
          ]
        }
      }
    }
  }
  ...
}
```

The AbstractRole property determines who has permission to call the API. Cloud Canvas has
built in Player, ProjectOwner, and DeploymentOwner roles. You can create others as required.
The abstract role specified here is mapped to an actual AWS IAM role using metadata on the role
definitions. For details, see Understanding the Resource Manager Security System (p. 2370).

The Action and ResourceSuffix are used by the Cloud Canvas Resource Manager access control to
generate an AWS IAM policy document statement. This process is described in Statement Reference of
IAM Policies for Executing API in API Gateway.

The Action property determines what the permission allows someone to do. When you grant
permissions to the Player abstract role, specify the permission execute-api:Invoke. This gives
the player permission to invoke the API. When you grant permissions to the ProjectOwner and
DeploymentOwner abstract roles, specify the permission execute-api:*. This grants permission to
invoke the API and manage the API's cache. For other roles, use your own requirements to determine
whether to grant a role permission to manage the cache, invoke the API, or both.

The access control system calculates the ARN of the API Gateway REST API resource to which the
action can be applied. The ResourceSuffix property from the Permission metadata provides
only the /stage-name/HTTP-VERB/resources-path-specifier part of the ARN described in the
Statement Reference of IAM Policies for Executing API in API Gateway.

For ServiceAPI, the stage-name is always api. You can specify either /api/... or /*/... in the
ResourceSuffix property value. A ResourceSuffix value of /api/* or /* grants permissions for all
operations on all paths of the service API.
Cloud Gem Framework Extension Object

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The swagger specification allows tools like the Cloud Gem Framework and API Gateway to define extension objects. These objects allow the `swagger.json` file to provide custom configuration data for the tool. The extension object that the Cloud Gem Framework uses is `x-amazon-cloud-canvas-lambda-dispatch`. This extension object simplifies the configuration of API Gateway for use with an AWS Lambda function.

You can place the `x-amazon-cloud-canvas-lambda-dispatch` object in any of the following swagger objects:

- **swagger object** – Sets the defaults for all paths and operations.
- **path item object** – Sets the defaults for all the path's operations and overrides the defaults set on the parent swagger object.
- **operation object** – Sets values for the operation and overrides the defaults set on the parent path and swagger objects.

The `x-amazon-cloud-canvas-lambda-dispatch` object supports the following properties:

- **lambda** – The ARN of the Lambda function that the operation invokes.
- **module** – The name of the module that defines the service function that processes the request.
- **function** – The name of the function that processes the request. The dispatch module in the Lambda function uses the `function` property and the `module` property to call your code when it receives a request. The dispatch module is described in the Upload Processing (p. 2318) section that follows.
- **additional-properties** – An object that provides properties that are added to the generated `x-amazon-apigateway-integration` object described in the next section.
- **additional-request-template-content** – A string that is inserted into the generated application/json request template, which is described in the next section.
- **additional-response-template-content** – An object that specifies additional content that is inserted into the generated application/json response template. This template is described in the next section. Properties named 200, 400, and 500 are supported. These correspond to the successful (200) and error (400 and 500) responses.

See the API Gateway documentation for more information about mapping templates.

Upload Processing

Before the `swagger.json` file is uploaded to API Gateway, the `x-amazon-cloud-canvas-lambda-dispatch` extension objects in the file are processed. This produces the `x-amazon-apigateway-integration` extension objects that configure API Gateway to call your cloud gem's AWS Lambda function.

The `x-amazon-cloud-canvas-lambda-dispatch` object and processing hide a lot of the complexity, and flexibility, of the `x-amazon-apigateway-integration` extension object. The Cloud Gems Framework provides a straightforward and recommended mapping to the API operations that are implemented in Lambda functions. API Gateway offers many other features that can be extremely useful when you must match an API that was implemented elsewhere or use API Gateway as a proxy for existing backend implementations.
You can still use all the power of the \texttt{x-amazon-apigateway-integration} object in your cloud gem API operations. If you include the object in the operation objects of your \texttt{swagger.json} file, the processing described here is skipped for the operation.

\textbf{Note}
You can also use the \texttt{lmbr\_aws\ cloud-gem-framework\ service-api-process-swagger} command to process the \texttt{swagger.json} file.

At minimum, the swagger object can include an \texttt{x-amazon-cloud-canvas-lambda-dispatch} object with a Lambda property. You can allow the module and function for each operation be determined automatically, or you can use \texttt{x-amazon-cloud-canvas-lambda-dispatch} objects to specify them.

The default \texttt{swagger.json} file provided by the Cloud Gem Framework contains the following \texttt{x-amazon-cloud-canvas-lambda-dispatch} object. The string \texttt{$ServiceLambdaArn$} is replaced with the \texttt{ServiceLambdaArn\ SwaggerSettings} property value provided in the \texttt{ServiceApi} resource definition.

\begin{verbatim}
"x-amazon-cloud-canvas-lambda-dispatch": {
  "lambda": "$ServiceLambdaArn$"
}
\end{verbatim}

When the \texttt{swagger.json} file is processed, an \texttt{x-amazon-apigateway-integration} object is added to every swagger operation object that does not have one. These objects have the following properties:

- \texttt{type} – Specify \texttt{AWS} to enable AWS Lambda function integration.
- \texttt{uri} – Construct the URI of the Lambda function with the value of the \texttt{x-amazon-cloud-canvas-lambda-dispatch} object's \texttt{lambda} property.
- \texttt{credentials} – The \texttt{Custom::ServiceApi} resource creates the ARN of a role. The role has a policy that is described in \textit{Access Control} (p. 2316).
- \texttt{requestTemplates} – A \texttt{application/json} template that causes a request as described in \textit{Default Request Mapping} (p. 2313). You can include additional content by using the \texttt{x-amazon-cloud-canvas-lambda-dispatch} object's \texttt{additional-request-template-content} property. This property can be used to pass other values, such as those defined by the \$context object, to the Lambda function. The additional request template content should start with a ', ' (and for consistent formatting, should start with ', \n ' and use ', \n ' between properties).
- \texttt{responses} Specifies – \texttt{application/json} templates for 200 (success), 400 (client error) and 500 (service error) responses as described in \textit{Default Response Mapping} (p. 2314). The 400 and 500 responses are inserted into the swagger responses object. However, you should define a 200 response that specifies a scheme that describes the data that the operation returns.
- Additional properties as specified by the \texttt{x-amazon-apigateway-integration-properties} property.

\textbf{Cross-Gem Communication}

\begin{verbatim}
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloud gems can use the cross-communication feature to expose their API operations to one another and use each other's backend services. For example, Lumberyard's \textit{Player Account Cloud Gem (p. 2188)} provides a banned player service that the \textit{Leaderboard Cloud Gem (p. 2188)} uses to limit fraudulent scores. You can also use cloud gem cross-communication to notify multiple gems when an event occurs.

To implement cross-gem communication, you define, implement, and use a Cloud Gem Framework \textit{Service API (p. 2305)} web service interface.
\end{verbatim}
Defining an Interface

To define a Cloud Gem Framework service interface, you use the swagger API definition (also known as the OpenAPI Specification) format to create a \interface-name\_interface-version\_json file. You then place the file in the \lumberyard\_version\dev\Gems\gem-name\AWS\api-definition directory. For an example in Lumberyard, see the \lumberyard\_version\dev\Gems \CloudGemPlayerAccount\AWS\api-definition\banplayer_1_0_0.json file.

The interface's full name has the format \gem-name\_interface-name\_interface-version. The \interface-version is a three part version number. The number follows the Semantic Versioning 2.0.0 standard but uses an underscore (_) separator instead of a period (.). When your interface definition changes, increment its version number according to this standard.

When the directory service maps available interface endpoints to client requests for endpoints, it takes the interface version into account. This enables clients that were built with an older minor version of an interface to use newer minor versions of the interface.

To define paths, operations, and data formats for the interface, you follow the swagger specification just as you would with any other swagger definition.

Implementing an Interface

To implement an interface, a gem's swagger.json file uses an x-cloud-gem-framework-interface-implementation extension object in a path object. The interface implementation object has the following properties:

- \interface – The full interface name (\gem-name\.interface-name\_interface-version) of the implemented interface.
- \module – The name of the Python module directory that contains the API implementation. If not provided, the full interface name is used. This directory contains the child modules with functions named as specified by the interface definition.

The referenced interface definition and the gem's swagger definition are merged before they are uploaded to configure Amazon API Gateway. The path objects in the interface definition are effectively inserted as child paths of the path object that defines the interface implementation extension object. The data definitions in each interface are given a unique prefix to prevent collisions with definitions from the gem's swagger definition or other interfaces.

For example, if the gem's swagger.json file contains the following:

```json
{
  ...
  "paths": {
    "/foo": {
      "x-cloud-gem-framework-interface-implementation": {
        "interface": "ExampleGem_ExampleInterface_1_0_0"
      }
    },
    "/bar": {
      "get": ...
    }
  }
}
```
"definitions": {
    "Bar": ...
  }
}

And the interface definition contains this:

{
  ...
  "paths": {
    "/aaa": { "get": { ... "$ref": "/definitions/ExampleData" ... } }  
    "/bbb": ... 
  },
  "definitions": {
    "ExampleData": ...
  }
}

The resulting swagger is something like the following:

{
  ...
  "paths": {
    "/foo/aaa": { "get": { ... "$ref": "ExampleGem_ExampleInterface_1_0_0.ExampleData" ... } }  
    "/foo/bbb": ... 
    "/bar": {
      "get": ...
    }
  },
  "definitions": {
    "Bar": ...
    "ExampleGem_ExampleInterface_1_0_0.ExampleData": ...
  }
}

Currently, you must manually implement the interface in the service code, although tooling that simplifies the implementation is planned for a future release.

To implement the interface, use the @service.api annotated methods that you use for other service API operations (p. 2314). To implement the API, the functions in your modules must use the names specified by the x-amazon-cloud-canvas-lambda-dispatch (p. 2318) extension object. To override the defaults, put x-amazon-cloud-canvas-lambda-dispatch extension objects that include module and function properties in your interface definition.

In the preceding example, extension objects were not used to override the defaults, so the code that implements the interface would have the following structure:

```python
api\  
  __init__.py  
bar.py  
  def get ...
foo_aaa.py  
  def get ...
foo_bbb.py  
```

**Note**
Currently, specifying the interface's path parameter inside the gem swagger file is not supported. Specify the path parameter in the interface swagger definition file instead.
Using an Interface

Calling the API operations that your interface defines from AWS Lambda functions is straightforward.

To use an Interface in a Lambda function

1. Specify the interface in the `Custom::LambdaConfiguration (p. 2256)` resource definition's `Services` property, as in the following example:

   ```json
   "ServiceLambdaConfiguration": {
     "Type": "Custom::LambdaConfiguration",
     "Properties": {
       "Services": [
         {
           "InterfaceId": "ExampleGem_ExampleInterface_1_0_0",
           "Optional": "False"
         },
         ...
       ],
       ...
     }
   },
   ...
   }
   ``

   The `Optional` property specifies whether the specified interface must exist. If `Optional` is `False` and no gem provides the specified interface, the deployment fails.

2. From your Lambda function code, get a client for the service. To do this:
   a. Use the `cfg_lambda_settings.get_service_url` function to get the service URL for the interface.
   b. Pass the URL to the `cfg_service_client.for_url` function, as in the following example:

   ```python
   import cfg_service_client
   import cfg_lambda_settings
   import boto3
   interface_url =
   cfg_lambda_settings.get_service_url("ExampleGem_ExampleInterface_1_0_0")
   service_client =
   cfg_service_client.for_url(interface_url, verbose=True,
   session=boto3._get_default_session())
   result =
   service_client.navigate('aaa').GET();
   ```

   Cloud Canvas ensures that the Lambda function has permission to call the service endpoints.

Game Clients

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Cloud Gem Framework can generate game clients for your service API.

The game client can call some API operations on behalf of the player. Other operations are callable only by other tools. By default, the client uses the AWS credentials by assuming the `Player` role through an Amazon Cognito identity pool. This process, described in detail in `Player Identity (p. 2379)`, is illustrated in the following diagram.
Generating a Game Client

Generating a game client is straightforward.

**To generate a game client**

Enter the following command:

```
lmbr_aws cloud-gem-framework generate-service-api-code --resource-group <group>
```

The `generate-service-api-code` command reads the `swagger.json` file that contains definitions for the resource group's service API. The command creates a C++ header file that implements a Lumberyard component. The header file is named `<group>ClientComponent.h`.

The directory location of the header file depends on whether `<group>` refers to a cloud gem or a project-defined resource group.

**Generated Component Header File Locations**

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Header File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloud gem</td>
<td>Gems&lt;gem&gt;\Code\AWS\ServiceApi</td>
</tr>
<tr>
<td>resource group</td>
<td>Code&lt;game&gt;\AWS&lt;group&gt;\ServiceAPI</td>
</tr>
</tbody>
</table>

**Component Header File Accessibility**

For a gem, the default location of the generated header file does not make it accessible outside of that gem. Typically, a cloud gem's service API is private to the gem, and the gem exposes its functionality through its own custom component. If you want to provide direct access to the cloud gem's API from other gems or game code, you can move the generated header file into the gem's `Code\Include` directory.
Notes

- To use the generated client, add the generated file to your gem or project's WAF file list, and then build the gem or project (p. 61).
- You can use the generated client component to invoke the API from C++ or Lua. For more information, see Calling a Game API (p. 2336).
- For sample code, see Generated Game Client Code Example (p. 2324).

Generated Game Client Code Example

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following client code was generated by the swagger.json file for the Message of the Day Gem that is included with Lumberyard. The swagger file for the gem defines many operations. For brevity, the example shows client code for only the following GET and PUT operations:

- GET /admin/messages?index={index}&count={count}&filter={filter}
- PUT /admin/messages/{msg_id}

For comments, see the Examining the Generated Game Client Code (p. 2335) section that follows the example.

```cpp
// Example Generated Game Client
#pragma once
#include <AzCore/Component/Component.h>
#include <AzCore/Component/Entity.h>
#include <AzCore/Component/ComponentBus.h>
#include <AzCore/EBus/EBus.h>
#include <AzCore/Memory/SystemAllocator.h>
#include <AzCore/Rtti/BehaviorContext.h>
#include <AzCore/Serialization/EditContext.h>
#include <AzCore/Serialization/SerializeContext.h>
#include <AzCore/std/smart_ptr/scoped_ptr.h>
#include <AzCore/std/string/conversions.h>
#if defined (PLATFORM_SUPPORTS_AWS_NATIVE_SDK)
#include <aws/core/http/HttpRequest.h>
#include <aws/core/http/HttpResponse.h>
#endif // (PLATFORM_SUPPORTS_AWS_NATIVE_SDK)
#include <LmbrAWS/ILmbrAWS.h>
#include <CloudGemFramework/ServiceRequestJob.h>
#include "StdAfx.h"

namespace CloudGemMessageOfTheDay {
namespace ServiceAPI {
    const char* LmbrAWS_CodeGen_PutAdminMessagesReturnType_UUID = "{ec889bb0-c329-11e6-b753-80a589a02a3d}";
    const char* LmbrAWS_CodeGen_Component_UUID = "{ec8874a1-c329-11e6-accd-80a589a02a3d}";
    const char* LmbrAWS_CodeGen_ResponseHandler_UUID = "{ec8874a4-c329-11e6-a067-80a589a02a3d}";
    const char* LmbrAWS_CodeGen_NotificationBus1_UUID = "{ec8874a2-c329-11e6-a661-80a589a02a3d}";
    const char* LmbrAWS_CodeGen_RequestBus1_UUID = "{ec8874a3-c329-11e6-a1e9-80a589a02a3d}";
```
const char* LmbrAWS_CodeGen_DeleteAdminMessagesReturnType_UUID= "{ec889bb1-c329-11e6-a94-80a589a02a3d}";
const char* LmbrAWS_CodeGen_DetailedMessageList_UUID= "{ec889bb4-c329-11e6-8650-80a589a02a3d}";
const char* LmbrAWS_CodeGen_ServiceStatus_UUID= "{45baaccf-c88b-11e6-b813-80a589a02a3d}";
const char* LmbrAWS_CodeGen_DetailedMessageData_UUID= "{ec889bb3-c329-11e6-bf48-80a589a02a3d}";

// redefs

bool WriteJson(CloudGemFramework::JsonWriter& writer, const int& item)
{
    return writer.Int(item);
}

bool WriteJson(CloudGemFramework::JsonWriter& writer, const AZStd::string& item)
{
    return writer.String(item);
}

bool WriteJson(CloudGemFramework::JsonWriter& writer, const float& item)
{
    return writer.Double(static_cast<double>(item));
}

struct PutAdminMessagesReturnType
{
    AZ_TYPE_INFO(PutAdminMessagesReturnType,
    LmbrAWS_CodeGen_PutAdminMessagesReturnType_UUID)
    AZ_CLASS_ALLOCATOR(PutAdminMessagesReturnType, AZ::SystemAllocator, 0)
    bool OnJsonKey(const char* key, CloudGemFramework::JsonReader& reader);
    static void Reflect(AZ::ReflectContext* reflection);
};

bool WriteJson(CloudGemFramework::JsonWriter& writer, const PutAdminMessagesReturnType& item)
{
    bool ok = true;
    ok = ok && writer.StartObject();
    ok = ok && writer.EndObject();
    return ok;
}

struct DetailedMessageData
{
    AZ_TYPE_INFO(DetailedMessageData, LmbrAWS_CodeGen_DetailedMessageData_UUID)
    AZ_CLASS_ALLOCATOR(DetailedMessageData, AZ::SystemAllocator, 0)

    int priority;
    AZStd::string message;
    AZStd::string endTime;
    AZStd::string UniqueMsgID;
    AZStd::string startTime;

    bool OnJsonKey(const char* key, CloudGemFramework::JsonReader& reader);
    static void Reflect(AZ::ReflectContext* reflection);
};

Version 1.28
2325
bool WriteJson(CloudGemFramework::JsonWriter& writer, const DetailedMessageData& item)
{
    bool ok = true;
    ok = ok && writer.StartObject();
    ok = ok && writer.Key("priority");
    ok = ok && WriteJson(writer, item.priority);
    ok = ok && writer.Key("message");
    ok = ok && WriteJson(writer, item.message);
    ok = ok && writer.Key("endTime");
    ok = ok && WriteJson(writer, item.endTime);
    ok = ok && writer.Key("UniqueMsgID");
    ok = ok && WriteJson(writer, item.UniqueMsgID);
    ok = ok && writer.Key("startTime");
    ok = ok && WriteJson(writer, item.startTime);
    ok = ok && writer.EndObject();
    return ok;
}

using DetailedMessageListPropertyList = AZStd::vector<DetailedMessageData>;

bool WriteJson(CloudGemFramework::JsonWriter& writer, const DetailedMessageListPropertyList& list)
{
    bool ok = true;
    ok = ok && writer.StartArray();
    for (auto item : list)
    {
        ok = ok && WriteJson(writer, item);
    }
    ok = ok && writer.EndArray();
    return ok;
}

struct DetailedMessageList
{
    AZ_TYPE_INFO(DetailedMessageList, LmbrAWS_CodeGen_DetailedMessageList_UUID)
    AZ_CLASS_ALLOCATOR(DetailedMessageList, AZ::SystemAllocator, 0)

    DetailedMessageListPropertyList list;

    bool OnJsonKey(const char* key, CloudGemFramework::JsonReader& reader);
    static void Reflect(AZ::ReflectContext* reflection);
};

bool WriteJson(CloudGemFramework::JsonWriter& writer, const DetailedMessageList& item)
{
    bool ok = true;
    ok = ok && writer.StartObject();
    ok = ok && writer.Key("list");
    ok = ok && WriteJson(writer, item.list);
    ok = ok && writer.EndObject();
    return ok;
}

// Service RequestJobs
CLOUD_GEM_SERVICE(CloudGemMessageOfTheDay);

void Configure()
{
    // Insert any necessary CloudGemFramework configuration here
}
class PutAdminMessagesRequest : public CloudGemFramework::ServiceRequest
{
    public:
        SERVICE_REQUEST(CloudGemMessageOfTheDay, HttpMethod::HTTP_PUT, "/admin/messages/{msg_id}");

        struct Parameters
        {
            AZStd::string msg_id;
            MessageData msg;

            bool BuildRequest(CloudGemFramework::RequestBuilder& request)
            {
                bool ok = true;

                ok = ok && request.SetPathParameter("{"msg_id"}", msg_id);

                ok = ok && request.WriteJsonBodyParameter(*this);
                return ok;
            }

            bool WriteJson(CloudGemFramework::JsonWriter& writer) const
            {
                bool ok = true;

                ok = ok && CloudGemMessageOfTheDay::ServiceAPI::WriteJson(writer, msg);
                return ok;
            }
        }

    PutAdminMessagesReturnType result;

    Parameters parameters;
};

using PutAdminMessagesRequestJob = CloudGemFramework::ServiceRequestJob<PutAdminMessagesRequest>;

class GetAdminMessagesRequest : public CloudGemFramework::ServiceRequest
{
    public:
        SERVICE_REQUEST(CloudGemMessageOfTheDay, HttpMethod::HTTP_GET, "/admin/messages");

        struct Parameters
        {
            int index;
            int count;
            AZStd::string filter;

            bool BuildRequest(CloudGemFramework::RequestBuilder& request)
            {
                bool ok = true;

                ok = ok && request.AddQueryParameter("index", index);
                return ok;
            }

            bool WriteJson(CloudGemFramework::JsonWriter& writer) const
            {
                bool ok = true;

                ok = ok && CloudGemMessageOfTheDay::ServiceAPI::WriteJson(writer, msg);
                return ok;
            }
        }

    GetAdminMessagesReturnType result;

    Parameters parameters;
};
ok = ok && request.AddParameter("count", count);
ok = ok && request.AddParameter("filter", filter);
ok = ok && request.WriteJsonBodyParameter(*this);
return ok;
}

bool WriteJson(CloudGemFramework::JsonWriter& writer) const
{
    bool ok = true;
    return ok;
}

DetailedMessageList result;
Parameters parameters;

using GetAdminMessagesRequestJob =
CloudGemFramework::ServiceRequestJob<GetAdminMessagesRequest>;

// Notification bus for this component
class CloudGemMessageOfTheDayNotifications
    : public AZ::ComponentBus
{
public:
    /**
     * Sent when the request is a success
     * 
     * Params:
     *    jsonOutput:       The output received from the Lambda call
     *    request:          The AWS Lambda request object
     */
    virtual void OnPutAdminMessagesRequestSuccess(const PutAdminMessagesReturnType response) {}

    /**
     * Sent when the request fails
     * 
     * Params:
     *    error:           The output received from the Lambda call; could be function error or an issue with the request
     *    request:         The AWS Lambda request object
     */
    virtual void OnPutAdminMessagesRequestError(const CloudGemFramework::Error error) {}

    /**
     * Sent when the request is a success
     * 
     * Params:
     *    jsonOutput:       The output received from the Lambda call
     *    request:          The AWS Lambda request object
     */
    virtual void OnGetAdminMessagesRequestSuccess(const DetailedMessageList response) {}

    /**
     * Sent when the request fails
     * 
     * Params:
     *    error:           The output received from the Lambda call; could be function error or an issue with the request
     */
    virtual void OnGetAdminMessagesRequestError(const CloudGemFramework::Error error) {}
* request: The AWS Lambda request object
*/
virtual void OnGetAdminMessagesRequestError(const CloudGemFramework::Error error)
{
};

using CloudGemMessageOfTheDayNotificationBus =
AZ::EBus<CloudGemMessageOfTheDayNotifications>;

class BehaviorCloudGemMessageOfTheDayNotificationBusHandler
 : public CloudGemMessageOfTheDayNotificationBus::Handler, public
AZ::BehaviorEBusHandler
{
public:
AZ_EBUS_BEHAVIOR_BINDER(BehaviorCloudGemMessageOfTheDayNotificationBusHandler,
LmbrAWS_CodeGen_NotificationBus1_UUID, AZ::SystemAllocator
, OnPutAdminMessagesRequestSuccess
, OnPutAdminMessagesRequestError
, OnGetAdminMessagesRequestSuccess
, OnGetAdminMessagesRequestError
);

void OnPutAdminMessagesRequestSuccess(const PutAdminMessagesReturnType response)
override
{
    Call(FN_OnPutAdminMessagesRequestSuccess, response);
}

void OnPutAdminMessagesRequestError(const CloudGemFramework::Error error) override
{
    Call(FN_OnPutAdminMessagesRequestError, error);
}

void OnPostAdminMessagesRequestSuccess(const DetailedMessageData response) override
{
    Call(FN_OnPostAdminMessagesRequestSuccess, response);
}

void OnPostAdminMessagesRequestError(const CloudGemFramework::Error error) override
{
    Call(FN_OnPostAdminMessagesRequestError, error);
}

};

class CloudGemMessageOfTheDayResponseHandler;

// Request bus for this component
class CloudGemMessageOfTheDayRequests
 : public AZ::ComponentBus
{

public:
virtual ~CloudGemMessageOfTheDayRequests() {}

virtual void PutAdminMessages(const AZStd::string& msg_id, const MessageData& msg,
CloudGemMessageOfTheDayResponseHandler* responseHandler) {}
virtual void GetAdminMessages(const int& index, const int& count, const
AZStd::string& filter, CloudGemMessageOfTheDayResponseHandler* responseHandler) {}

};

using CloudGemMessageOfTheDayRequestBus = AZ::EBus<CloudGemMessageOfTheDayRequests>;
// This class is used as a parameter for all requests and throws the response on the
CloudGemMessageOfTheDayNotificationBus
// Inherit from this to define custom behavior.
class CloudGemMessageOfTheDayResponseHandler
{
  public:
    AZ_TYPE_INFO(CloudGemMessageOfTheDayResponseHandler,
      LmbrAWS_CodeGen_ResponseHandler_UUID)
    AZ_CLASS_ALLOCATOR(CloudGemMessageOfTheDayResponseHandler, AZ::SystemAllocator, 0)
    virtual ~CloudGemMessageOfTheDayResponseHandler() {}

    virtual void HandlePutAdminMessagesSuccess(PutAdminMessagesRequestJob* job,
      AZ::Entity* entity)
    {
      EBUS_EVENT_ID(entity->GetId(), CloudGemMessageOfTheDayNotificationBus,
        OnPutAdminMessagesRequestSuccess, job->result);
    }

    virtual void HandlePutAdminMessagesError(PutAdminMessagesRequestJob* job,
      AZ::Entity* entity)
    {
      EBUS_EVENT_ID(entity->GetId(), CloudGemMessageOfTheDayNotificationBus,
        OnPutAdminMessagesRequestError, job->error);
    }

    virtual void HandleGetAdminMessagesSuccess(GetAdminMessagesRequestJob* job,
      AZ::Entity* entity)
    {
      EBUS_EVENT_ID(entity->GetId(), CloudGemMessageOfTheDayNotificationBus,
        OnGetAdminMessagesRequestSuccess, job->result);
    }

    virtual void HandleGetAdminMessagesError(GetAdminMessagesRequestJob* job,
      AZ::Entity* entity)
    {
      EBUS_EVENT_ID(entity->GetId(), CloudGemMessageOfTheDayNotificationBus,
        OnGetAdminMessagesRequestError, job->error);
    }
};

class CloudGemMessageOfTheDayClientComponent
  : public AZ::Component
  , public CloudGemMessageOfTheDayRequestBus::Handler
{
  public:
    AZ_COMPONENT(CloudGemMessageOfTheDayClientComponent,
      LmbrAWS_CodeGen_Component_UUID);
    virtual ~CloudGemMessageOfTheDayClientComponent() = default;

    AZStd::scoped_ptr<CloudGemMessageOfTheDayResponseHandler> m_defaultResponseHandler;

  void Init() override
  {
    m_defaultResponseHandler.reset(new CloudGemMessageOfTheDayResponseHandler());
    CloudGemMessageOfTheDay::ServiceAPI::Configure();
  }

  void Activate() override
  {
    CloudGemMessageOfTheDayRequestBus::Handler::BusConnect(m_entity->GetId());
  }

  void Deactivate() override
  { Version 1.28
2330 }
CloudGemMessageOfTheDayRequestBus::Handler::BusDisconnect();

static void Reflect(AZ::ReflectContext* reflection)
{
    MessageData::Reflect(reflection);
    PutAdminMessagesReturnType::Reflect(reflection);
    DetailedMessageData::Reflect(reflection);
    DetailedMessageList::Reflect(reflection);
    MessageList::Reflect(reflection);

    AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(reflection);
    if (serializeContext)
    {
        // we must include any fields we want to expose to the editor or lua in the
        // serialize context
        serializeContext->Class<CloudGemMessageOfTheDayClientComponent>("
            CloudGemMessageOfTheDay Component"
        )->Version(1);
        AZ::EditContext* editContext = serializeContext->GetEditContext();
        if (editContext)
        {
            editContext->Class<CloudGemMessageOfTheDayClientComponent>("CloudGemMessageOfTheDayClientComponent",
                "CloudGemMessageOfTheDay Component")
                ->Attribute(AZ::Edit::Attributes::Category, "Cloud Canvas
                Gems")
                ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                AZ_CRC("Game"));
        }
    }

    AZ::BehaviorContext* behaviorContext = azrtti_cast<AZ::BehaviorContext*>(reflection);
    if (behaviorContext)
    {
        behaviorContext->EBus<CloudGemMessageOfTheDayRequestBus>("CloudGemMessageOfTheDayRequestBus")
            // one of these for each function
            ->Event("PutAdminMessages",
            &CloudGemMessageOfTheDayRequestBus::Events::PutAdminMessages)
            ->Event("GetAdminMessages",
            &CloudGemMessageOfTheDayRequestBus::Events::GetAdminMessages);
        behaviorContext->EBus<CloudGemMessageOfTheDayNotificationBus>("CloudGemMessageOfTheDayNotificationBus")
            ->Handler<BehaviorCloudGemMessageOfTheDayNotificationBusHandler>();
    }
}

// Functions from the swagger definitions

void PutAdminMessages(const AZStd::string& msg_id, const MessageData& msg, CloudGemMessageOfTheDayResponseHandler* responseHandler) override
{
    if (responseHandler == nullptr)
    {
        responseHandler = AZStd::get_pointer(m_defaultResponseHandler);
    }
// create job
PutAdminMessagesRequestJob* job = PutAdminMessagesRequestJob::Create(
    [responseHandler, this](PutAdminMessagesRequestJob* job)
    {
        // handle success
        responseHandler->HandlePutAdminMessagesSuccess(job, m_entity);
    },
    [responseHandler, this](PutAdminMessagesRequestJob* job)
    {
        // handle error
        responseHandler->HandlePutAdminMessagesError(job, m_entity);
    });
job->parameters.msg = msg;
job->parameters.msg_id = msg_id;
job->Start();

void GetAdminMessages(const int& index, const int& count, const AZStd::string& filter, CloudGemMessageOfTheDayResponseHandler* responseHandler) override
{
    if (responseHandler == nullptr)
    {
        responseHandler = AZStd::get_pointer(m_defaultResponseHandler);
    }

    // create job
    GetAdminMessagesRequestJob* job = GetAdminMessagesRequestJob::Create(
        [responseHandler, this](GetAdminMessagesRequestJob* job)
        {
            // handle success
            responseHandler->HandleGetAdminMessagesSuccess(job, m_entity);
        },
        [responseHandler, this](GetAdminMessagesRequestJob* job)
        {
            // handle error
            responseHandler->HandleGetAdminMessagesError(job, m_entity);
        });
    job->parameters.index = index;
    job->parameters.count = count;
    job->parameters.filter = filter;
    job->Start();
}

bool MessageData::OnJsonKey(const char* key, CloudGemFramework::JsonReader& reader)
{
    if (strcmp(key, "priority") == 0) return reader.Accept(priority);
    if (strcmp(key, "message") == 0) return reader.Accept(message);
    if (strcmp(key, "endTime") == 0) return reader.Accept(endTime);
    if (strcmp(key, "startTime") == 0) return reader.Accept(startTime);
    return reader.Ignore();
```cpp
void MessageData::Reflect(AZ::ReflectContext* reflection)
{
    AZ::SerializeContext* serializeContext =
    azrtti_cast<AZ::SerializeContext*>(reflection);
    if (serializeContext)
    {
        serializeContext->Class<MessageData>()
            ->Version(1);
    }
    AZ::BehaviorContext* behaviorContext =
    azrtti_cast<AZ::BehaviorContext*>(reflection);
    if (behaviorContext)
    { }
    AZ::BehaviorContext* behaviorContext =
    azrtti_cast<AZ::BehaviorContext*>(reflection);
    if (behaviorContext)
    { }
}

bool PutAdminMessagesReturnType::OnJsonKey(const char* key,
CloudGemFramework::JsonReader& reader)
{
    return reader.Ignore();
}

void PutAdminMessagesReturnType::Reflect(AZ::ReflectContext* reflection)
{
    AZ::SerializeContext* serializeContext =
    azrtti_cast<AZ::SerializeContext*>(reflection);
    if (serializeContext)
    {
        serializeContext->Class<PutAdminMessagesReturnType>()
            ->Version(1);
    }
    AZ::BehaviorContext* behaviorContext =
    azrtti_cast<AZ::BehaviorContext*>(reflection);
    if (behaviorContext)
    { }
}

bool DetailedMessageData::OnJsonKey(const char* key, CloudGemFramework::JsonReader& reader)
{
    return true;
}
```
if (strcmp(key, "priority") == 0) return reader.Accept(priority);
if (strcmp(key, "message") == 0) return reader.Accept(message);
if (strcmp(key, "endTime") == 0) return reader.Accept(endTime);
if (strcmp(key, "UniqueMsgID") == 0) return reader.Accept(UniqueMsgID);
if (strcmp(key, "startTime") == 0) return reader.Accept(startTime);
return reader.Ignore();
}

void DetailedMessageData::Reflect(AZ::ReflectContext* reflection)
{
    AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(reflection);
    if (serializeContext)
    {
        serializeContext->Class<DetailedMessageData>()->Version(1);
    }

    AZ::BehaviorContext* behaviorContext = azrtti_cast<AZ::BehaviorContext*>(reflection);
    if (behaviorContext)
    {
        behaviorContext->Class<DetailedMessageData>("CloudGemMessageOfTheDay_DetailedMessageData")
            ->Attribute(AZ::Script::Attributes::Storage,
                        AZ::Script::Attributes::StorageType::Value)
            ->Property("priority",
                        BehaviorValueProperty(&DetailedMessageData::priority))
            ->Property("message", BehaviorValueProperty(&DetailedMessageData::message))
            ->Property("endTime", BehaviorValueProperty(&DetailedMessageData::endTime))
            ->Property("UniqueMsgID", BehaviorValueProperty(&DetailedMessageData::UniqueMsgID))
            ->Property("startTime", BehaviorValueProperty(&DetailedMessageData::startTime));
    }
}

bool DetailedMessageList::OnJsonKey(const char* key, CloudGemFramework::JsonReader& reader)
{
    if (strcmp(key, "list") == 0) return reader.Accept(list);
    return reader.Ignore();
}

void DetailedMessageList::Reflect(AZ::ReflectContext* reflection)
{
    AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(reflection);
    if (serializeContext)
    {
        serializeContext->Class<DetailedMessageList>("CloudGemMessageOfTheDay_DetailedMessageList")
            ->Attribute(AZ::Script::Attributes::Storage,
                        AZ::Script::Attributes::StorageType::Value)
Examining the Generated Game Client Code

The following table provides additional information for the generated game client code example in the previous section. The line numbers indicate the location in the code that the comments refer to.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24, 25</td>
<td>The C++ namespace for the API. This is always <code>&lt;resource-group-name&gt;::ServiceApi</code>.</td>
</tr>
<tr>
<td>56, 75, 110, 124</td>
<td>The <code>PutAdminMessageReturnType</code> struct contains the PUT operation's return data as defined by the swagger file. In this case, it is an empty object and has no properties.</td>
</tr>
<tr>
<td>67, 92, 112, 137</td>
<td>The <code>WriteJson</code> method is generated to handle the serialization of <code>PutAdminMessageReturnType</code> objects to JSON text format. JSON format is required for the data to be sent to the service. A similar function is generated for each of the type definitions in the <code>swagger.json</code> file.</td>
</tr>
<tr>
<td>150</td>
<td>The <code>CLOUD_GEM_SERVICE</code> macro defines a class that provides information that is common to all of the service's requests.</td>
</tr>
<tr>
<td>152</td>
<td>The <code>Configure</code> function is called when the generated component is initialized. You can add code to this function to change the default configuration for the service. For example, the following code increases the timeout for <code>GetServiceStatus</code> requests.</td>
</tr>
<tr>
<td></td>
<td>```cpp</td>
</tr>
<tr>
<td></td>
<td>void Configure()</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>GetServiceStatusRequestJob::GetDefaultConfig()-&gt;requestTimeoutMs = 20000;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>If you regenerate the client, your changes to the <code>Configure</code> method are lost.</td>
</tr>
<tr>
<td>157, 202</td>
<td>A <code>Request</code> class such as <code>PutAdminMessagesRequest</code> is generated for each of the operations defined in the <code>swagger.json</code> file. This class encapsulates the HTTP method and path that are used to make the request. The <code>Parameters</code> struct in the class defines the fields for each of the request's parameters that are specified by the swagger file.</td>
</tr>
<tr>
<td>200, 240</td>
<td>The Lumberyard job system executes API requests asynchronously. This work is performed by the <code>CloudGemFramework::ServiceRequestJob</code> class and specialized by the <code>PutAdminMessagesRequest</code> class declared previously. The C++ <code>using</code> statement creates an alias for that type.</td>
</tr>
</tbody>
</table>
Calling a Game API

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To call your API operations, you can use C++, the client component, a request job, or Lua.

Topics

- Calling an API from C++ (p. 2336)
- Using Lua to Call an API (p. 2339)

Calling an API from C++

To invoke a service API from C++, you can use an EBus event handler from the generated client component. You can also use the generated service request job class directly. If you want to make requests from a component, consider using the client component. If you want to perform a sequence of operations, consider using the job class.

Using the Client Component to Call an API

The following example from the CloudGemLeaderboard service API shows a component that invokes GetServiceStatusRequestJob. There are two ways to receive the request's response:

1. Use an object that inherits from the service's response handler.
2. Connect to the generated component's notification bus.

The example shows both methods.
// Sample API Caller
#include <AzCore/Component/Component.h>
#include <AzCore/Component/Entity.h>
#include <AzCore/Serialization/EditContext.h>
#include <AzCore/Serialization/SerializeContext.h>
#include <AzCore/EBus/EBus.h>

#include <AWS/ServiceAPI/CloudGemLeaderboardClientComponent.h>
#include <CloudGemLeaderboard/CloudGemLeaderboardBus.h>
class SampleResponseHandler :
    public CloudGemLeaderboard::ServiceAPI::CloudGemLeaderboardResponseHandler
{
public:
    void HandleGetServiceStatusSuccess(CloudGemLeaderboard::ServiceAPI::GetServiceStatusRequestJob* job, AZ::Entity* entity) override
    {
        // Look at job->result for response data
        AZ_Printf("Got response: %s", job->result.status.c_str());
    }

    void HandleGetServiceStatusError(CloudGemLeaderboard::ServiceAPI::GetServiceStatusRequestJob* job, AZ::Entity* entity) override
    {
        // Look at job->error for error data
    }
};

class SampleAPICallerComponent :
    public AZ::Component,
    public CloudGemLeaderboard::ServiceAPI::CloudGemLeaderboardNotificationBus::Handler
{
public:
    SampleResponseHandler* m_responseHandler;
    AZ_COMPONENT(SampleAPICallerComponent, "{aedd6408-e2f0-4250-a181-b0ef41085a94}");
    virtual ~SampleAPICallerComponent() = default;
    static void Reflect(AZ::ReflectContext* reflection)
    {
        AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(reflection);
        if (serializeContext)
        {
            // We must include any fields we want to expose to the editor or Lua in the serialize context
            serializeContext->Class<SampleAPICallerComponent>()
                ->Version(1);

            AZ::EditContext* editContext = serializeContext->GetEditContext();
            if (editContext)
            {
                editContext->Class<SampleAPICallerComponent>("SampleAPICallerComponent",
                    "Component to call CloudGemLeaderboard GetServiceStatus")
                    ->ClassElement(AZ::Edit::ClassElements::EditorData,"
                        ->Attribute(AZ::Edit::Attributes::AppearsInAddComponentMenu,
                          AZ_CRC("Game"));
                }
            }
        }
    }

    void Activate() override
    {
        CloudGemLeaderboard::ServiceAPI::CloudGemLeaderboardNotificationBus::Handler::BusConnect(m_entity->GetId());
    }
Using the Request Job to Call an API

In some situations you might want more control over how your background jobs are executed. For example, you might want to change the thread pool that certain requests use or perform multiple requests on a background thread. You can use the generated request job classes directly to make these changes.

In its simplest form, you can use the request job class in the following example. This is the same code that is in line 452 of the generated request component.

```cpp
PutAdminMessagesRequestJob* job = PutAdminMessagesRequestJob::Create(
    [responseHandler, this](PutAdminMessagesRequestJob* job)
    {
        // Handle success
        responseHandler->HandlePutAdminMessagesSuccess(job, m_entity);
    },
    [responseHandler, this](PutAdminMessagesRequestJob* job)
    {
        // Handle error
        responseHandler->HandlePutAdminMessagesError(job, m_entity);
    });

job->parameters.msg = msg;
job->parameters.msg_id = msg_id;
job->Start();
```
To learn about other ways to use the request job class, explore the `ServiceApiRequestJob` class definition and see *Running AWS API Jobs Using the Cloud Gem Framework* (p. 2286).

**Using Lua to Call an API**

The following code example shows how to call a service API from a Lua script.

```lua
-- Service status getter
local leaderboardstatusgetter = {
    Properties = {
    }
}

function leaderboardstatusgetter:OnActivate()
    self.notificationHandler = CloudGemLeaderboardNotificationBus.Connect(self, self.entityId)
    CloudGemLeaderboardRequestBus.Event.GetServiceStatus(self.entityId, nil)
end

function leaderboardstatusgetter:OnDeactivate()
    self.notificationHandler:Disconnect()
end

function leaderboardstatusgetter:OnGetServiceStatusRequestSuccess(response)
    Debug.Log(response.status)
end

function leaderboardstatusgetter:OnGetServiceStatusRequestError(error)
    Debug.Log(error.message)
end

return leaderboardstatusgetter
```

**Publishing Your API**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Service API operations are implemented by API Gateway, which has an open source serverless web application that you can use to create your own developer portal. A developer portal on API Gateway empowers you to do the following:

- List your API operations in catalog form.
- Allow programmers to sign up.
- Display documentation that helps programmers understand your API.
- Let programmers test your API and provide feedback.
- Grow a programmer ecosystem.
- Monetize your API and grow API product revenue.

For more information, see Generate Your Own API Gateway Developer Portal.

**Using the Cloud Gem Framework Command Line**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use the `lmbr_aws cloud-gem-framework` command line for working with Cloud Canvas Cloud Gems (p. 2127) and the Cloud Gem Framework Service API (p. 2305).

**Note**
For general information on the `lmbr_aws` command line tool, including common syntax, configuration, and arguments, see Using the Cloud Canvas Command Line (p. 2390).

**Commands**

Following are details of the `lmbr_aws cloud-gem-framework` commands.

**add-service-api-resources**

Adds the resources to implement a `swagger.json` file-based service API to a resource group.

The `add-service-api-resources` command accepts the following arguments:

- `--resource-group {resource-group-name}` or `-r {resource-group-name}`
  - Required. The name of the resource group or cloud gem.
- `--force`
  - Optional. Forces the replacement of existing resource definitions. By default, existing resource definitions with the same names are not changed.

For more information about the `add-service-api-resources` command, see Resources (p. 2311) in the Cloud Gem Framework Service API (p. 2305) documentation and Getting Started with the Cloud Gem Framework (p. 2258).

**generate-service-api-code**

Generates component and Lambda function code to support a service API that is described in a gem or resource group `swagger.json` file. The generated code is created in subdirectories under the `\dev\Gems\<gem-name>\Code` directory.

The `generate-service-api-code` command accepts the following arguments:

- `--gem {gem-path}` or `-g {gem-path}`, `--resource-group {resource-group-path}` or `-r {resource-group-path}`
  - Required. The path of a cloud gem or resource group subdirectory that provides the `swagger.json` file that is used to generate the code (for example, `C:\lumberyard_version\dev\Gems\MyCloudGem\v1`).
- `--component-client-path {component-client-path}`
  - Optional. The output path for the component client code files `<gem-name>ClientComponent.h` and `<gem-name>ClientComponent.cpp`. The default location is the `\dev\Gems\{cloud-gem-name\}\VN\Code\AWS\ServiceApi` directory.
- `--update-waf-files`
  - Optional. Adds the generated `.h` and `.cpp` code files to the `.waf_files` file that is in the gem's `\dev\Gems\<gem name>\Code` subdirectory. The `.waf_files` file is a manifest file for the Waf build of the gem.

The following sample `.waf_files` file is for a gem called MyCloudGem. Its full path is `\dev\Gems\MyCloudGem\Code\mycloudgem.waf_files`. 

```
For more information about the `generate-service-api-code` command, see Generating a Game Client (p. 2323) and Getting Started with the Cloud Gem Framework (p. 2258) For more information about Waf, see Using the Waf Build System (p. 63).

**remove-service-api-resources**

Remove the resources that implement a `swagger.json` file-based service API operation from a resource group.

The `remove-service-api-resources` command accepts the following arguments:

- `--resource-group {resource-group-name}` or `-r {resource-group-name}`
  
  Required. The name of the resource group.

**service-api-process-swagger**

Processes the Cloud Canvas extension objects that are defined in a `swagger.json` file and produces swagger definitions that can be imported into Amazon API Gateway.

The `service-api-process-swagger` command accepts the following arguments:

- `--resource-group {resource-group-name}` or `-r {resource-group-name}`
  
  Required. The name of the resource group.

- `--input {file-path}`
  
  Optional. The file from which the `swagger` JSON is read. For the cloud gems that are included with Lumberyard, the default is `dev/Gems/{gem-name}/swagger.json`.

- `--output {file-path}`
  
  Optional. The file to which the processed `swagger` JSON is written. By default, the output is written to `stdout` (the command line console window).

For more information about the `service-api-process-swagger` command, see Cloud Gem Framework Extension Object (p. 2318).
Using Shared Code

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The `lmbr_aws cloud-gem-framework add-service-api-resources (p. 2340)` command adds a service API to a resource group. Prior to Lumberyard version 1.10, it copied the service Lambda code for dispatching service API calls from the

```
CloudGemFramework\AWS\resource-manager-code
\default-resource-group-content\lambda-function-code directory to the resource group's
lambda-function-code directory.
```

In version 1.10, Lumberyard adds a general purpose code sharing mechanism. You can use this mechanism to include a single copy of the service API dispatch code in all the Lambda functions that require it. Place the code to be shared in a subdirectory of a gem's `AWS\common-code` directory. The subdirectory name is the name of the code package.

An `.import` file can be found in a gem's `AWS` directory or a project's resource group directory. The file can be placed any of the following directories to indicate that the code depends on a set of specified packages:

- `lambda-code\<lambda-name>\`
- `lambda-function-code`
- `<lambda-name>-lambda-code`
- `resource-manager-code`
- `common-code\<package-name>\`

You can use the `common-code\<package-name>\` directory to add dependencies recursively. The `lambda-code\<lambda-name>` directory is new in Lumberyard 1.10. For more information about this directory, see Lambda Code Directories (p. 2240).

The `.import` file has the following format:

```
<gem-name>\<package-name>
<gem-name>\<package-name>
...
```

The entries in the `.import` file instruct the Cloud Gem framework to include the contents of the specified gem's `AWS\common-code\<package-name>` directory. For Lambda code, the contents of the directory are included in the `.zip` file that contains other Lambda code. When resource manager loads the code specified by `sys.path` (the Python default module search path, or `PYTHONPATH`), the `resource-manager-code` directory is included in that path.

The service API dispatch code can be found at `Gems\CloudGemFramework\v<N>\AWS\common-code\LambdaService`. The `Gems\CloudGemFramework\v<N>\AWS\common-code\LambdaSettings` directory contains code for accessing injected settings.

Cloud Gem Framework and Resource Manager Versioning

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Lumberyard offers a versioning system that makes it easier to update a project from one version of the Cloud Gem Framework and the **Cloud Canvas Resource Manager** to another. The versioning system has the following advantages:

- Cloud gems can have separate versions.
- Cloud gems can specify the version of Lumberyard that they work with.
- The resource manager and Cloud Gem Framework are implemented in directory structures that support versioning.
- Lumberyard can provide multiple versions of the Cloud Gem Framework and the resource manager at the same time.
- Lumberyard can innovate while still supporting gems that depend on early versions of the framework.

This document describes these changes at an architectural level. For concrete steps on updating existing projects and gems to use versioning, see Updating Projects and Cloud Gems to Version 1.0.0 of the Cloud Gem Framework (p. 2345).

**Versioning Convention**

Lumberyard gems use a simplified semantic versioning, which defines version numbers in the form `major.minor.revision`. Lumberyard increments the major version for changes that cause previous code not to work, including any change that breaks the code or configuration that is not directly controlled by the Cloud Gem Framework.

Every major version change is released by creating a new code base in a new `\dev\Gems\CloudGemFramework\vN` directory, where `N` is the major version number.

Minor updates replace the code in the `\dev\Gems\CloudGemFramework\vN` directory. Earlier major versions of the framework continue to be released with Lumberyard for an unspecified amount of time before they are removed from the release.

**Multiple Versions of the Cloud Gem Framework Gem**

The new directory structure in Cloud Gem Framework 1.0.0 enables coexistence of multiple versions of the Cloud Gem Framework Gem. Gem implementation has moved from the `\dev\Gems\CloudGemFramework\` directory to the `\dev\Gems\CloudGemFramework\v<N>` directory. The intent is that all nonbreaking changes be made in the `v<Current>` directory. When breaking changes occur, a `v<Next>` directory is created to contain the updated implementation. The `v<Current>` directory contents remain either unchanged or updated without breaking changes. A gem can now have different versions of its `gem.json` file in the `v1` and the `v2` directory, for example. This enables the gem to specify different versions. Lumberyard's build and project configuration tools support gems in subdirectories.

**Note**

The **Cloud Canvas Resource Manager** now allows gems to be in directories other than `\Gems\<gem-name>`. The `resource-group add` (p. 2408) `lmbr_aws resource-group add` command's `--gem` option now takes an optional value that specifies the gem directory path. The specified directory path can be either relative to the current working directory or a full path.

**Applying Framework Updates to a Project**

When Lumberyard releases a new major version of the framework, you can choose when to disable the old framework version and when to enable the new one.

If a minor version of the framework is released and you replace the framework configuration in the `CloudGemFramework\vN` directory, errors will occur. The errors are displayed in the console when you load your project into Lumberyard Editor.

After the new version is discovered and enabled, the next step is to update the project's infrastructure in AWS as dictated by the new framework version. The `lmbr_aws` tool and the Cloud Canvas Resource
Manager detect when an upgrade is needed by checking the project's current framework version in two places: the `local-project-settings.json` file and in the project's Amazon S3 configuration bucket. If either value is not exactly the same as the framework version, the command exits with an error and takes no action.

To update your project infrastructure, use the `lmbr_aws project update-framework-version` command. If the `update-framework-version` command detects a framework version change, it performs the following actions:

1. Executes the `before_project_framework_version_change(hook, from_version, to_version)` hook method in plugin update.py modules.
2. Updates the project stack.
3. Executes the `after_project_framework_version_change(hook, from_version, to_version)` hook method in plugin update.py modules.
4. Saves the new framework version to the `local-project-settings.json` file and to the project configuration bucket in Amazon S3.

The project's framework version is updated after all hooks are successfully called and all updates completed.

**Update Deployments Manually**

The `lmbr_aws project update-framework-version` command never updates deployment, deployment access, or resource group stacks. The update hooks can make changes to resource templates, Lambda code, and other items. However, you must perform all deployment, deployment access and resource group stack updates separately after the `project update-framework-version` command completes.

To help custom tools deal with old stacks, the framework version that is in effect for the stack is provided by the `FrameworkVersion` template parameter. If this parameter is not present, the tool should assume that the stack predates version 1.1.0 of the framework.

In the case of a major version change, Lumberyard tools (including `lmbr_aws`) can refuse to work with any stack that has not yet been updated. However, for minor version changes, the tools should continue to work with deployment and resource group stacks that have not yet been updated.

To update the deployment stack (and all of its resource group stacks), you can use the `lmbr_aws deployment update (p. 2397)` command. To update the deployment access stack, you can use the `deployment update-access (p. 2397)` command.

**Resource Manager Merged into the Cloud Gem Framework Gem**

In Lumberyard 1.10, resource manager functionality has been taken over by the Cloud Gem Framework Gem. Accordingly, and to allow resource manager to be versioned, the contents of the \dev\tools \lmbr_aws\ directory have moved to the dev\Gems\CloudGemFramework\v<N>\ResourceManager directory, with the following exceptions:

- The \dev\tools\lmbr_aws\ directory still contains the cli.py and gui.py modules. These modules are loaded by the `lmbr_aws.cmd` file and the resource manager user interface in Lumberyard Editor. These modules discover which project is current by looking in the \dev \bootstrap.cfg file. They then look in the project's \dev\project_name\gems.json file to get the `Version` value for CloudGemFramework. The modules then forward the request to the corresponding cli.py or gui.py module for the specified version of the framework. If no version of the framework is enabled, a warning message that the gem must be enabled is displayed.
- The dev\Tools\lmbr_aws\test\ directory still contains the RunAllTests.cmd, cleanup.cmd, and Python module files that support them. The RunAllTests.cmd file has been updated to run tests from the CloudGemFramework\v<N>\ directory. As new releases occur, the file will be updated to include all versions of the framework.
Global Project Code Directories and Project Templates

In Cloud Gem Framework 1.0.0 (Lumberyard version 1.10), the project code directories and project templates have also changed to support versioning.

Project Code Directories

The `<project>`\AWS\project-code directory formerly contained code for the following Lambda functions:

- ProjectPlayerAccessTokenExchangeHandler
- ProjectResourceHandler
- ProjectServiceLambda

Previously, this Lambda code was copied from the \dev\tools\lmbr_aws\AWSResourceManager\default-project-content\project-code directory when the project was created.

This code is now located in the Gems\CloudGemFramework\vN\AWS\lambda-code directory in subdirectories divided by Lambda function.

**Note**

Using a `<gem>`\AWS\project-code or a `resource-group`\<resource-group>\project-code directory to inject code into the project Lambda function is no longer supported.

Project Templates

The following template files have moved to the Gems\CloudGemFramework\vN\ResourceManager\resource_manager\templates directory.

- deployment-access-template.json
- deployment-template.json
- project-template.json

When the framework updates a stack, it uses these templates as a base to create the project's actual templates. Then it uploads the templates to AWS CloudFormation.

**Note**

Starting in Lumberyard 1.10, you can use extension files to add resources to each of these templates. For more information, see Template Extension Files (p. 2239).

Sharing Code

The lmbr_aws cloud-gem-framework add-service-api-resources (p. 2340) command adds a service API to a resource group. Before Lumberyard version 1.10, it copied the service Lambda code for dispatching service API calls from the CloudGemFramework\AWS\resource-manager-code \default-resource-group-content\lambda-function-code directory to the resource group's lambda-function-code directory.

In version 1.10, Lumberyard adds a general purpose code sharing mechanism. You can use this mechanism to include a single copy of the service API dispatch code in all the Lambda functions that require it. For more information, see Using Shared Code (p. 2342).

Updating Projects and Cloud Gems to Version 1.0.0 of the Cloud Gem Framework

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
A cloud gem is a gem that depends on the Cloud Gem Framework (p. 2257). Starting in Lumberyard 1.10 (and Cloud Gem Framework 1.0.0), gem versioning support (p. 2342) makes the work of updating Lumberyard cloud gem projects much easier. However, manual steps are required to update projects and gems that were created before Cloud Gem Framework version 1.0.0.

**Updating Existing Cloud Gem Projects**

To update an existing cloud gem project, you must update the project's gems.json file and runtime configuration.

**To update an existing Lumberyard project**

1. In the project's dev\<project-name>\gems.json file, find the entry for CloudGemFramework. Change the Version value to 1.0.0 and the Path value to Gems/CloudGemFramework/v1, as in the following example.

   ```json
   {
     "Path": "Gems/CloudGemFramework/v1",
     "Uuid": "6fc787a982184217a5a553ca24676cfa",
     "Version": "1.0.0",
     "_comment": "CloudGemFramework"
   },
   ```

2. In a command prompt window, run the following command to update the project's runtime configuration.

   ```cmd
   dev\Tools\LmbrSetup\Win\lmbr.exe projects populate-appdescriptors -projects <project-name>
   ```

   For more information about populate-appdescriptors, see Projects Commands (p. 58) in the documentation for Managing Game Projects with Lmbr.exe (p. 56).

**Update the Project's AWS Directory**

To update the project's AWS directory, you must delete the project, deployment access, and deployment template files. But before you delete the template files, you must save any roles or resources that you added.

**To update the AWS directory**

1. Delete the <project>\AWS\project-code directory. If you have made changes to this code, you must update the CloudGemFramework\\Wm\lambda-code directory content to preserve the changes that you made.

2. If you added roles or other resources to the <project>\AWS\project-template.json file, create a <project>\AWS\project-template-extensions.json file and copy those resource definitions into the project-template-extensions.json file.

3. Delete the <project>\AWS\project-template.json file.

4. If you added roles or other resources to the <project>\AWS\deployment-access-template.json file, create a <project>\AWS\deployment-access-template-extensions.json file and copy those resource definitions into the deployment-access-template-extensions.json file.

5. Delete the <project>\AWS\deployment-access-template.json file.

6. If you added resources to the <project>\AWS\deployment-template.json file, create a <project>\AWS\deployment-template-extensions.json file and copy those resource definitions into the deployment-template-extensions.json file.
7. Delete the `<project>`\AWS\deployment-template.json file.

**Update the Project's Resource Group Directories**

When no Lambda resources exist for a resource group, you can simply remove the resource group's Lambda code directories.

**To remove Lambda code directories from a project resource group**

1. For each `<project>`\AWS\resource-group\<resource-group-name> directory, check whether its resource-template.json file defines AWS Lambda function resources.
2. If the resource-template.json file does not define any Lambda resources, delete the following directories:
   - `<project>`\AWS\resource-group\<resource-group-name>\lambda-function-code
   - `<project>`\AWS\resource-group\<resource-group-name>\<lambda-name>-lambda-code
3. If the resource-template.json file defines AWS Lambda function resources, perform the steps described in Updating Lambda Code (p. 2347).

**Updating Existing Cloud Gems**

To update an existing cloud gem, you must update the gem's gem.json file and either remove its Lambda code directories or update its Lambda code.

**To update an existing cloud gem**

1. In the `<gem>`\gem.json file, find the entry for CloudGemFramework. Change VersionConstraints to the following value.

   ```json
   "VersionConstraints": [ "~>1.0" ],
   ```

2. If the `<gem>`\AWS\resource-template.json file does not exist or does not define any AWS Lambda function resources, delete the `<gem>`\AWS\lambda-function-code directory and any `<gem>`\AWS\<lambda-name>-lambda-code directories. Otherwise, perform the steps in Updating Lambda Code (p. 2347).

**Updating Lambda Code**

Updating Lambda code involves reorganizing your existing Lambda code directories and creating .import files as needed. For information on the reasons for this directory restructuring, see Cloud Gem Framework and Resource Manager Versioning (p. 2342).

**To update your Lambda code**

1. For each of the following Lambda code directories, perform the steps that follow.
   - `<project>`\AWS\resource-group\<resource-group-name>\lambda-function-code
   - `<project>`\AWS\resource-group\<resource-group-name>\<lambda-name>-lambda-code
   - `<gem>`\AWS\lambda-function-code
   - `<gem>`\AWS\<lambda-name>-lambda-code
2. If the code directory contains the service.py and errors.py files that support service API dispatching, delete them and add an .import file with the following content to the directory:
3. If the code directory contains a CloudCanvas subdirectory, delete the subdirectory. If you did not have to create an .import file in step 2, add an .import file that has the following content:

```java
CloudGemFramework.LambdaService
```

If you already created an .import file in step 2, add CloudGemFramework.LambdaSettings to the .import file, as in the following example:

```java
CloudGemFramework.LambdaService
CloudGemFramework.LambdaSettings
```

4. Move the contents of the code directory (along with the new .import file, if any) into a `<parent-dir>\lambda-code\<lambda-name>` directory. Note the following:

- If you had code for multiple Lambda functions in the single `lambda-function-code` directory, decide which modules to put into the subdirectory of the `lambda-code` directory, and place them there.
- If you have code that is used by multiple Lambda functions, put that code in a `<parent-dir>\common-code\<import-name>` directory. To include the code with the rest of the Lambda code when it is uploaded, use an .import file.

For more information about these changes, see Cloud Gem Framework and Resource Manager Versioning (p. 2342).

## Administering Cloud Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/lumberyard/) or visit the AWS Game Tech blog to learn more.

This section provides information for administering your Lumberyard project's AWS account and its use of the AWS cloud.

### Topics

- Setting Up a Project to Use Resource Manager (p. 2348)
- Working with Deployments (p. 2352)
- Freeing Up AWS Resources (p. 2361)
- Understanding the Resource Manager Security System (p. 2370)

## Setting Up a Project to Use Resource Manager

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/lumberyard/) or visit the AWS Game Tech blog to learn more.

After initializing Cloud Canvas Resource Manager so that your Amazon Lumberyard project can use AWS, you can use Resource Manager to configure the stacks for your project.

### Topics
• Initializing Cloud Canvas Resource Manager (p. 2349)
• Working with Project Stacks (p. 2350)

Initializing Cloud Canvas Resource Manager

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you perform an operation that requires an AWS account, and no account has been associated with your Lumberyard project, the Initialize Cloud Canvas Resource Manager dialog prompts you for the required information.

To initialize Cloud Canvas Resource Manager

1. When prompted to initialize the Cloud Canvas Resource Manager, provide the following information:

   • For Project stack name, enter the name of an AWS CloudFormation stack that you will create. The stack will contain the AWS resources that Cloud Canvas Resource Manager will use for your project. By default, Lumberyard uses the name of your project for the stack name. A stack with the name that you specify must not already exist in your AWS account for the region you select.

   • For AWS Credentials, select from the list of available profiles or create a new one. If you have no AWS profiles on your computer, you are prompted for an AWS secret key and an AWS access key. You can also edit an existing one.

   In order to use Lumberyard with AWS, you must provide administrative credentials for your AWS account either directly, or through an AWS profile. For information on how to get these credentials from AWS, see the Tutorial: Getting Started with Cloud Canvas (p. 2116).

   • For AWS region, specify the AWS data center where your resources will reside. You must choose a region that supports all the AWS services that your game uses. The region you choose must also
support the Amazon Cognito service, which Lumberyard uses to establish player identity, and AWS CloudFormation, which Lumberyard uses to create and manage resources. For more information about the capabilities of different regions, see AWS Regions and Endpoints.

2. Click Create to start the initialization process. In the navigation tree, the Project stack (p. 2350) node is selected, and in the detail pane, the Progress log (p. 2228) shows the progress of the initialization.

## Working with Project Stacks

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you select the Project stack node in the Cloud Canvas Resource Manager navigation pane, the detail pane shows information about the AWS CloudFormation stack that Cloud Canvas is using.

![Cloud Canvas Resource Manager](image)

Note the following:

- If you select a project stack node and no AWS profile is configured, Lumberyard prompts you to provide one. A profile is required for Lumberyard to display the status of your project's resources. For more information, see Managing Cloud Canvas Profiles (p. 2223).
- If you select the Project stack node when the project has not been initialized for use with Cloud Canvas, Lumberyard prompts you to initialize the project and create a project stack. For more information, see Initializing Cloud Canvas Resource Manager (p. 2349).

### Project Stack Status Table

The Project stack status table shows the status of the AWS CloudFormation stack that contains the resources used by your project's resource groups.

This table has the following columns:

- **Pending** – Indicates when changes have been made to the local stack templates or code but not yet uploaded to AWS. Pausing your mouse pointer on a value in this column displays the reason why the change is pending. Following are the possible values:
- **Create** – The stack is defined locally but has not been created in AWS.
- **Update** – The stack's template and/or Lambda function code has been changed locally but not yet uploaded to AWS.
- **Delete** – The stack has been removed from the local definitions but has not been deleted in AWS. If you are using a source control system, this status can indicate that a new stack has been added to the project. However, your local copy of the source control is out of date and doesn't contain the definition for the new stack.
- **--** – No change is pending.

**Status** – The status of the AWS CloudFormation stack. See Understanding Resource Status Descriptions (p. 2225) for a description of the values this column can have. To see additional status information, pause your pointer on the status indicator.

**Created** – The time the stack was created.

**Updated** – The time the stack status was updated.

**ID** – A truncated version of the AWS ID for the stack. To see the full ID, pause your pointer on the truncated ID.

**Upload Resources**

Click **Upload resources** to start the process of modifying, creating, or deleting resources in AWS so that they match your local definitions of them. A dialog box shows the changes that will be applied.

The table of changes has the same columns as the stack resource status table but also includes a column for **Impacts**. If a change has the potential to affect the security of your project, the **Impacts** column contains the text **Security**. Before you can start an operation that has a security impact, you must select the option **It is OK that this will impact resource SECURITY**. Review any security changes carefully before you apply them.

Similarly, before you can start an operation that deletes one or more resources, you must select the option **It is OK that this will permanently DELETE resources**.

**Stack Resources Table**

The **Stack resources** table shows the status of the resources that your project is using.
This table has the following columns:

**Pending** – Indicates that changes have been made locally but not yet uploaded to AWS. Pausing your mouse on a value in this column displays the reason why the change is pending. Following are the possible values:

- **Create** – The resource is defined locally but has not been created in AWS.
- **Update** – The resource has been changed locally but not yet uploaded to AWS.
- **Delete** – The resource has been removed from the local definitions but has not been deleted on AWS.
  If you are using a source control system, this status can indicate that a new resource has been added to the project. However, your local copy of the source control is out of date and doesn't contain the definition for the new resource.
- **--** – No change is pending.

**Resource Name** – The logical name of the resource.

**Type** – The type of the resource (for example, a Lambda function, Amazon S3 bucket, or a custom resource).

**Status** – The current condition of the resource. For a description of the possible status values, see **Understanding Resource Status Descriptions (p. 2225)**. To see additional status information, pause your pointer on the status.

**Timestamp** – The time of the most recent change.

**ID** - A truncated version of the AWS ID for the stack. To see the full ID, pause your pointer on the truncated ID.

**Working with Deployments**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Topics**

- **Create Deployment** (p. 2353)
- **Deployment Status Table** (p. 2354)
- **Individual Deployment Nodes** (p. 2354)
- **Making a Cloud Canvas Deployment Active** (p. 2355)
- **Testing Different Mappings** (p. 2357)
- **Using Protected Deployments** (p. 2359)
- **Deleting Cloud Canvas Deployments and Their Resources** (p. 2360)

A deployment is an independent copy of the AWS resources that your game uses. Deployments are useful for maintaining a safe separation among game lifecycle phases such as development, test, and production. In the resource manager navigation pane, the **Deployments** node shows you the status of your project’s deployments. You can also use it to create a new deployment.
Note: If the **Deployments** node is selected when no AWS profile is configured, Lumberyard prompts you to provide a profile. The status of the project's deployments cannot be displayed unless a profile is provided. For more information, see [Managing Cloud Canvas Profiles](#) (p. 2223).

**Create Deployment**

Click **Create deployment** to start the creation of a deployment:

When uploading resources for the first time, you may see this version of the dialog:

Provide a name for **Deployment name**. Lumberyard appends this name to the project stack name to create an AWS CloudFormation stack for the deployment.
To start the deployment creation process, click OK. In the resource manager navigation pane, a node for the deployment appears under Deployments. In the detail pane, the Viewing the Cloud Canvas Progress Log (p. 2228) provides details about the creation process.

**Deployment Status Table**

The Deployment status table shows the status of the AWS CloudFormation stack for each deployment. Deployment shows the deployment name. For descriptions of the remaining fields in this table, see Stack Resources Table (p. 2351) in the Working with Project Stacks (p. 2350) section.

**Individual Deployment Nodes**

The child nodes of the Deployment node each represent one of the Lumberyard project's deployments. When a Deployment node is selected, the detail pane shows the current status of the selected deployment.

![Deployment Status Table](image)

**Note**

If a Deployment node is selected when no AWS profile is configured, Lumberyard prompts you to provide a profile. The status of the project’s deployments cannot be displayed unless a profile is provided. For more information, see Managing Cloud Canvas Profiles (p. 2223).

**Individual Deployment Status Table**

The <Deployment Name> deployment status table shows the status of the AWS CloudFormation stack for the selected deployment. For descriptions of the contents of this table, see Project Stack Status Table (p. 2350) in the Working with Project Stacks (p. 2350) section.

**Upload All Resources**

Click Upload all resources to start the process of modifying, creating, or deleting resources in the current AWS deployment so that they match your local definitions for all resource groups.
Delete Deployment

Click **Delete deployment** to start the process of deleting the deployment's resources from AWS. The resources defined by all resource groups will be deleted.

For more information about deleting deployments, see Deleting Cloud Canvas Deployments and Their Resources (p. 2360).

Stack Resources Table

The **Stack resources** table shows the status of each of the resources defined by all the resource groups for the selected deployment. For descriptions of the fields in this table, see Stack Resources Table (p. 2351) in the Working with Project Stacks (p. 2350) section.

Making a Cloud Canvas Deployment Active

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can select the deployment that you want Lumberyard Editor to consider active. The active deployment is the deployment that you work with in Lumberyard Editor. Lumberyard Editor uses the active deployment's resources when you launch your game. When you select the Working with Resource Groups (p. 2228) node or an Managing Individual Resource Groups (p. 2231) node in the Cloud Canvas Resource Manager navigation pane, the status information that appears corresponds to the active deployment.

You can also select the deployment that you want to be active by default for all team members.

**Note**

To select a deployment, you must have initialized Cloud Canvas Resource Manager to work with your AWS account and created a deployment. For more information, see Initializing Cloud Canvas Resource Manager (p. 2349) and Create Deployment (p. 2353).

Making a Deployment Active

You have several ways to make a deployment active in Cloud Canvas Resource Manager.

**To make a deployment active**

- To make a deployment active, do one of the following:
  - In Lumberyard Editor, click AWS, **Cloud Canvas, Select a deployment**.
• In the **Cloud Canvas Resource Manager** toolbar, click the name of the current deployment, or click **(none)** if none is configured:

![Current deployment: (none)](image)

When prompted, choose the deployment that you want to make active:

![Select deployment](image)

One or more of the deployments may be marked **protected**. For more information, see **Using Protected Deployments** (p. 2359).

• In the **Cloud Canvas Resource Manager** navigation pane, right-click the deployment that you want to make active, and then click **Make active deployment**:

![Deployments](image)

### Making a Deployment the Default

You can use the **Cloud Canvas Resource Manager** to make a deployment the default.

**To make a deployment active by default for all team members**

1. In Lumberyard Editor, click **AWS, Cloud Canvas, Cloud Canvas Resource Manager**.
2. In the **Cloud Canvas configuration** navigation tree, expand **Administration (advanced)**, and then expand **Deployments**.

3. Right-click the deployment that you want to make the default, and then click **Make default deployment**:

![Image of Cloud Canvas configuration tree with deployments expanded]

To use the command line to make a deployment the default

- To use the command line to make a deployment the default, enter the following command:

```
lmbr_aws deployment default --set <deployment name>
```

### Testing Different Mappings

**Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.**

**Download O3DE** or visit the **AWS Game Tech blog** to learn more.

To test your client with different Cloud Canvas resource deployments, you can export mappings by using Cloud Canvas Resource Manager or the **lmbr_aws** command line.

**Note**

As of Lumberyard 1.11, separate mappings are exported for the player game client and game server to differentiate the resources that are visible to each.

**To export mappings from Cloud Canvas Resource Manager**

- In Resource Manager, do one of the following:
  - Left click a deployment and click **Export Mapping** in the main window.
  - Right click the name of a deployment name the list, and then select **Export Mapping** from the context menu.

The following image shows both options.
To export a mapping from the command line

- Enter the following command, where `<name>` is the name of your deployment.

```
lmbr_aws mappings update --deployment <name>
```

The mapping files for the specified deployment are created in the `\dev\<project_name\>Config` directory and have the format `<deployment_name>.player.awsLogicalMappings.json` and `<deployment_name>.server.awsLogicalMappings.json`.

**Tip**

Using the command line to export mappings makes it easy for you to create scripts for testing or development.

**Selecting a Deployment with a PC Launcher**

After you have exported one or more mappings, you can choose the mapping to use when you run a game launcher such as the one at `dev\Bin64\vcNN\CloudGemSamples\CloudCanvas\CloudGemSamplesLauncher.exe`.

To direct the launcher to use a specific deployment, use the command line option `cc_override_resource_map`, as in the following example.
CloudGemSamplesLauncher.exe -cc_override_resource_map Config\dev.player.awsLogicalMappings.json

The argument for the `cc_override_resource_map` parameter specifies the mapping file that you want to use.

If you have exported a single mapping file to the launcher, the launcher uses that mapping file by default. If you have exported multiple mapping files to the launcher, you must select a mapping by using the `cc_override_resource_map` parameter. If you don't specify a mapping after multiple mappings have been exported, the launcher gives an error message, and no mapping is loaded.

Using Protected Deployments

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Cloud Canvas to mark specific deployments as protected. Protected status makes it more difficult for users like programmers, testers, or game developers to inadvertently connect a development game client to live resources.

When a user starts a protected game, a message box notifies the user that he or she is attempting to use a protected deployment. The user is given the option to not connect before any potentially harmful data is transmitted.

The protection feature purposely uses a message box that "breaks" automation. If the scripts that run tests are configured to use a protected deployment, the Lumberyard client will not continue without human intervention.

When Protected Deployments Are Detected

When a game is run from Lumberyard Editor, protection is always detected. When a game is run from a Windows launcher, protection is detected only when the launcher is running in debug mode.

Marking a Deployment as Protected

Currently, you must set the protection from the `lmbr_aws` command line tool by using the `protect-deployment` command.

The `protect-deployment` command uses the following parameters.

- `--set <deployment_name>` – Specifies that the deployment is protected.
- `--clear <deployment_name>` – Specifies the deployment is not protected.
- `--show` - Displays a list of currently protected deployments.

To display the protected status of deployments, you can also use either the `list-deployments` or `list-mappings` command.

Viewing Protected Status in Cloud Canvas Resource Manager

In Cloud Canvas Resource Manager, you can view, but not change, the status of protected deployments. The ability to change the protected status of deployments from Lumberyard Editor is planned for a future release.

Note

Setting a deployment to protected does not prevent you from deploying or deleting resources by using Cloud Canvas Resource Manager or the `lmbr_aws` command line tool; it only enables
the warning functionality. For this reason, be careful not to make unnecessary changes to critical deployments. A more comprehensive model for protecting deployments is planned for a future version of Lumberyard.

**Deleting Cloud Canvas Deployments and Their Resources**

Delete Cloud Canvas Deployments and Their Resources.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To remove Cloud Canvas functionality from your Lumberyard project and the AWS resources related to it, you can use Cloud Canvas Resource Manager or the Cloud Canvas command line.

**Warning**

Only administrators should perform these actions. If you remove all AWS resources managed by Cloud Canvas for your Lumberyard project, the players of your game will not be able to access any of the Cloud Canvas resource groups that implement your game's cloud connected features.

**To use Cloud Canvas Resource Manager to delete Cloud Canvas deployments and their resources**

1. If you have checked Lumberyard into source control, ensure that the `<root>\<game>\AWS\project-settings.json` file has been checked out and is writeable.
2. In Lumberyard Editor, choose AWS, Cloud Canvas, Cloud Canvas Resource Manager.
3. In the Cloud Canvas configuration navigation pane, expand Administration (advanced), and then expand Deployments. The list of the deployments in the project appears.
4. Select the deployment to delete and click Delete deployment.
5. When prompted to confirm, click Yes to start the process of deleting the deployment's resources from AWS. The process might take a few minutes.
6. To remove all of the project's resources from AWS, follow the same steps to delete each of the project's deployments.

**To use the command line to delete Cloud Canvas deployments and their resources**

1. If you have checked Lumberyard into source control, ensure that the `<root>\<game>\AWS\project-settings.json` file has been checked out and is writeable.
2. Open a command line prompt and change to your the Lumberyard `dev` directory.
3. Determine the project's deployment names by entering the following command:
4. Enter the following command for each of the deployments that you want to delete:

```
lmbr_aws deployment delete --deployment <deployment name>
```

**Note**
To remove all Cloud Canvas functionality from your project, use the `delete-deployment` command to delete all of deployments that were listed by `list-deployments`. Then remove the project stack as described in the step that follows.

5. After you have deleted all deployments, you can delete the resources that Cloud Canvas uses to manage your project by entering the following command:

```
lmbr_aws project delete
```

This removes all AWS resources that are related to your Cloud Canvas project.

**Freeing Up AWS Resources**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After a time of intensively testing or developing Lumberyard game projects that use Cloud Canvas, your attempts to create resources in AWS might fail or result in rollback loops. For example, you might receive error messages like the following:

```
CREATE FAILED for Configuration (AWS::S3::Bucket with ID "bucket_id"). You have attempted to create more buckets than allowed.
```

```
CREATE FAILED for ServiceApi (Custom::ServiceApi with ID "api_id"). Failed to create resource. Unexpected RuntimeError occurred: [...] An error occurred (BadRequestException) when calling the ImportRestApi operation: Maximum number of API operations has been reached. Please contact AWS if you need additional API operations.
```

These errors occur when you exceed limits that some AWS services have on the number of resources that you can create per account. The number of resources allowed per account varies by service and by resource. The following table shows some limits for some AWS resources that are commonly used with Cloud Canvas:

<table>
<thead>
<tr>
<th>AWS Service</th>
<th>Limit</th>
<th>Information Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon S3</td>
<td>100 buckets per account.</td>
<td>Amazon Simple Storage Service (Amazon S3) Limits</td>
</tr>
<tr>
<td>Amazon API Gateway</td>
<td>60 APIs per account per region.</td>
<td>Amazon API Gateway Limits, Pricing and Known Issues</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
<td>Initially, 256 tables per region.</td>
<td>Limits in DynamoDB</td>
</tr>
</tbody>
</table>
For more information on the resource limits of individual AWS services, see the AWS Service Limits in the AWS General Reference.

If necessary, you can contact AWS customer service to increase the resource limits for some services. However, if you have resources that you no longer require, it might be easier to simply remove them from your account.

To remove unused resources from AWS, the following tools are available:

- The Cloud Canvas Cleanup Tool (p. 2363)
- The AWS Command Line Interface (CLI) (p. 2366)
- The AWS Management Console (p. 2369)

Of these, the Cloud Canvas cleanup tool provides the quickest way to delete resources.

The Cloud Canvas Cleanup Tool

The Lumberyard installation includes the Cloud Canvas cleanup tool. You can use the Cloud Canvas cleanup tool to delete AWS resources from your account that have the prefix that you specify. The tool is located in the \lumberyard_version\dev\Tools\lmbr_aws\test directory.

Prerequisites

To use the cleanup tool, you must complete the following:

- Have access to a Windows computer.
- Install Lumberyard on your computer.
- Set up and configure an AWS admininstrator IAM profile and set an admininstrator profile name as default on your computer.

For more information, see Step 2: Create an IAM User to Administer the Cloud Canvas Project (p. 2117) and Step 4: Add Administrator Credentials to Lumberyard (p. 2119) in the Cloud Canvas tutorial (p. 2116).

- Install the AWS CLI, configure it with an admininstrator IAM profile, and set it to your preferred region.

For instructions on how to install the AWS CLI on Windows, see Installing the AWS Command Line Interface. The AWS CLI requires Python. You may use the version of Python distributed as part of Lumberyard, provided that it's available in your system's PATH.

- Use Cloud Canvas Resource Manager or the lmbr_aws tool to create a Cloud Canvas project stack.

  **Note**
  If you are using a fresh installation of Lumberyard and need to clean up resources before you can create a project stack, perform the following steps.

  **To use the cleanup tool without creating a project stack**

  1. In Project Configurator, set the default project to a project that has the Cloud Canvas Common gem enabled.
  2. Determine the name of the AWS profile that you want to use. To discover existing profiles, you can examine either the credentials or config file in the %SystemDrive% \Users\user_name\.aws\ directory.
  3. Enter the following command to configure the AWS CLI to use the profile.

```bash
aws configure --profile profile_name
```

  4. In the lumberyard_installation\dev directory, run the following command.
lmbr_aws profile default --set profile_name

This adds the profile to the DefaultProfile section of the user-settings.json (p. 2237) file. As of Lumberyard version 1.15, the Cloud Canvas cleanup tool requires that a default profile be set in the user-settings.json file.

Example

The following example command sets the default Cloud Canvas profile to CloudCanvasAdmin for the CloudGemDefectReportSample project in the Lumberyard installation location C:\Lumberyard\.

C:\Lumberyard\dev>lmbr_aws profile default --set CloudCanvasAdmin

The command produces the following output.

[WAF] Engine Root: C:\Lumberyard\dev\ Saving C:\Lumberyard\dev\Cache\CloudGemDefectReportSample\pc\User\AWS\user-settings.json Default Profile: CloudCanvasAdmin

Using the Cloud Canvas Cleanup Tool

Before you use the cleanup tool, be aware of the following points:

- Do not use the cleanup tool if you have a project stack name that begins with an IAM user name that you do not want to delete. Doing so can result in the deletion of the IAM user, its roles, and its profiles.
- When you delete an AWS resource, you permanently delete any objects that are stored in that resource. For example, if you delete an S3 bucket, all objects inside the bucket are also deleted.

To use the Cloud Canvas cleanup tool

1. Open a command prompt window.
2. Navigate to the lumberyard_version\dev directory.
3. Determine the string prefix (full or partial) that you want to specify to delete resources. For more information, see Identifying Cloud Canvas Prefixes (p. 2364).
4. Enter the following command:
   
   `lmbr_aws --cleanup --prefix prefix_string`

   For example, the following command removes AWS resources that begin with the prefix cloudgemssamples.

   `lmbr_aws --cleanup --prefix cloudgemssamples`

   Depending on the number of resources, the command might take some time to complete.
5. Repeat the command until the cleanup tool no longer finds items to delete. Multiple runs can be required. The cleanup tool attempts to delete resources as they become available for deletion. However, because the cleanup tool attempts to delete resources in type order, some resources are not available for deletion until after the script finishes.
The `--cleanup` argument will take priority over other `lmbr_aws` arguments.

**Note**
The `lmbr_aws --cleanup` command runs the same script as the `cleanup` command script found in the `lumberyard_version\dev\Tools\lmbr_aws` directory. Either one can be used with the same list of optional arguments described in the list that follows.

### Optional Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--aws-access-key</code></td>
<td>The AWS access key to use.</td>
</tr>
<tr>
<td><code>--aws-secret-key</code></td>
<td>The AWS secret access key to use.</td>
</tr>
<tr>
<td><code>--delete-global-resources</code></td>
<td>If the <code>--region</code> argument is specified, deletes global resources such as IAM roles and Amazon S3 buckets. If the <code>--region</code> argument is not specified, the <code>--delete-global-resources</code> argument is ignored. This argument is new in Lumberyard version 1.16.</td>
</tr>
<tr>
<td><code>--except</code> <code>exception [exception ...]</code></td>
<td>Do not delete resources that start with the prefixes specified.</td>
</tr>
<tr>
<td><code>-h, --help</code></td>
<td>Shows a help message and exits.</td>
</tr>
<tr>
<td><code>--prefix</code> <code>prefix [prefix ...]</code></td>
<td>Deletes stacks and Amazon S3 buckets in AWS that have the specified prefixes.</td>
</tr>
<tr>
<td><code>--profile</code> <code>profile</code></td>
<td>The AWS profile to use. Defaults to the default AWS profile.</td>
</tr>
<tr>
<td><code>--region</code> <code>region</code></td>
<td>The AWS region to use. Defaults to <code>us-east-1</code>.</td>
</tr>
</tbody>
</table>

### Identifying Cloud Canvas Prefixes

Use the Cloud Canvas resource naming conventions in this section to identify the resources that you want to delete. In general, the AWS Management Console uses the names that you provided for your resources when you created them in Cloud Canvas Resource Manager or `lmbr_aws`.

#### Project Stacks

In Cloud Canvas, project stack names correspond to AWS CloudFormation stack names. For example, if you create a Cloud Canvas project stack called `CloudGemSamples`, the project stack appears in the AWS CloudFormation Console as `CloudGemSamples`.

#### Project Stack Resources

Resources in a Cloud Canvas project stack have the following form.

```
ProjectStackName-ResourceName
```

For example, if a `CloudGemSamples` Cloud Canvas project stack has an Amazon S3 bucket named `Storage`, the bucket appears in the Amazon S3 Console as `CloudGemSamples-Storage`.

#### Deployment Stacks

A deployment stack name in AWS has the following form.

```
ProjectStackName-DeploymentStackName
```
For example, if a CloudGemSamples Cloud Canvas project stack has a deployment stack named TestDeployment, the deployment stack appears in the AWS CloudFormation Console as CloudGemSamples-TestDeployment.

**Deployment Stack Resources**

Cloud Canvas deployment stack resource names have the following form.

```
ProjectStackName-DeploymentName-ResourceGroupOrGemName-ResourceName
```

For example, suppose a CloudGemSamples Cloud Canvas project stack has a deployment stack named TestDeployment. If TestDeployment has a cloud gem named TestGem that uses an Amazon S3 bucket named GemResourceBucket, the bucket appears in the Amazon S3 Console as CloudGemSamples-TestDeployment-TestGem-GemResourceBucket.

**Viewing Prefixes in the AWS Management Console**

You can use the AWS Management Console to identify resources to delete.

To view AWS CloudFormation stacks

2. In the upper-right corner of the management console, choose the AWS region from which you want to delete resources.
3. Identify the AWS CloudFormation stack(s) that you no longer require.
4. Use the Cloud Canvas resource naming rules to determine the prefix to use with the Cloud Canvas cleanup tool.

To identify orphaned resource(s)

1. Open the AWS Console for the resource that you believe is orphaned (for example, the Amazon S3 Console or the Amazon API Gateway Console).
2. In the upper-right corner of the management console, choose the AWS region from which you want to delete resources.
3. Identify a resource that you no longer require.
4. Use the Cloud Canvas resource naming rules to determine the prefix to use with the Cloud Canvas cleanup tool.

**Notes Regarding the Cleanup Tool**

When you use the Cloud Canvas cleanup tool, note the following.

- You do not have to specify a complete prefix. The cleanup tool takes the string that you specify and matches it with all resources that have names that start with that string. For example, if you specify the prefix Cloud, all resources that start with Cloud (like CloudGemSamples) are deleted, including resources that AWS CloudFormation created.
- An Amazon S3 bucket/aws_resource can seem to fail to delete even though it is empty and visible in the management console.

A resource that has been removed can have an obsolete (ghost) reference to it in the AWS Management Console. If you attempt to use the cleanup tool or AWS CLI to delete the resource, you receive an error that the resource does not exist. For example, when you try to delete an S3 bucket that has already been deleted, an error message like the following occurs:
ERROR An error occurred (NoSuchBucket) when calling the ListObjectVersion operation: The specified bucket does not exist.

This error confirms that the resource has been deleted. The resource will no longer count against your resource limits or costs. AWS automatically cleans up the ghost reference in 24 to 72 hours.

- Errors can occur when you delete a Cloud Canvas project stack prefix for your current Lumberyard project and then use Cloud Canvas Resource Manager or \lmbr_aws\ commands. This is because the resources that the project expects no longer exist in AWS.

To resolve this issue, you can delete the \lumberyard_version\dev\project_name\AWS\local-project-settings.json file, which is automatically regenerated by Cloud Canvas Resource Manager and \lmbr_aws\ commands. Alternatively, you can edit the ProjectStackId section of the local-project-settings.json file to remove the resources listed that have been deleted. You should also edit the Mappings section of the \lumberyard_version\dev\Cache\game\OS\user\AWS\user-settings.json file to remove references to resources that have been deleted.

AWS Command Line Interface (CLI)

You can use the AWS Command Line Interface to remove resources from specific AWS services in your account. This section lists some useful commands for the following AWS services that are commonly used with Cloud Canvas:

- Amazon DynamoDB (p. 2366)
- AWS Lambda (p. 2367)
- Amazon API Gateway (p. 2368)
- Amazon S3 (p. 2368)

Prerequisites

To use the AWS CLI, you must complete the following:

- Set up and configure an AWS administrator IAM profile and set an admininstrator profile name as default on your computer.

  For more information, see Step 2: Create an IAM User to Administer the Cloud Canvas Project (p. 2117) and Step 4: Add Administrator Credentials to Lumberyard (p. 2119) in the Cloud Canvas tutorial (p. 2116).

- Install the AWS CLI, configure it with an admininstrator IAM profile, and set it to your preferred region.

  For instructions on how to install the AWS CLI on Windows, Linux, macOS, or Unix, see Installing the AWS Command Line Interface. The AWS CLI requires Python. You may use the version of Python distributed as part of Lumberyard, provided that it's available in your system's PATH.

Creating Script Files for Batch Deletion

A generally useful ad hoc strategy is to use a service-specific AWS CLI command to redirect a list of the resources into a text file. You can then use the text file programatically or convert the file into a shell script of AWS CLI deletion commands. The next few sections illustrate this technique.

Amazon DynamoDB

To show list of the DynamoDB tables in the current region in a command prompt window, enter the following command.
aws dynamodb list-tables

To specify a different region, use the --region region_name argument, as in the following example.

aws --region us-west-2 dynamodb list-tables

To redirect the list of tables into a text file, enter a command like the following.

aws dynamodb list-tables >ddb_table_list.txt

The example command redirects the output into a text file called ddb_table_list.txt.

The AWS CLI command to delete a DynamoDB table uses the following syntax.

aws dynamodb delete-table --table-name table_name

To create a shell script that deletes many or all tables at once, you can edit the text file that has the list-tables output. Use a search and replace operation to precede each table name with the aws dynamodb delete-table command. Do another search and replace to remove the double quote and comma at the end of each line. The result looks like the following example.

aws dynamodb delete-table --table-name CloudGemSamples-CGSamplesDeployment-CloudGemDynamicContent-hash-StagingSettingsTable
aws dynamodb delete-table --table-name CloudGemSamples-CGSamplesDeployment-CloudGemInGameSurvey-hash-AnswerAggregations
aws dynamodb delete-table --table-name CloudGemSamples-CGSamplesDeployment-CloudGemInGameSurvey-hash-Questions
aws dynamodb delete-table --table-name CloudGemSamples-CGSamplesDeployment-CloudGemInGameSurvey-hash-Surveys
aws dynamodb delete-table --table-name CloudGemSamples-CGSamplesDeployment-CloudGemLeaderboard-hash-BannedPlayerTable

After you rename the file with a .cmd or .bat extension (Windows), you can run the script to delete all the tables in one go.

For more information on the DynamoDB CLI commands, see dynamodb.

AWS Lambda

To list AWS Lambda functions, enter the following command.

aws lambda list-functions

The following syntax deletes a Lambda function.

aws lambda delete-function --function-name function_name

The deletion syntax requires only the function name, so to create a script, filter the JSON output of the list-functions command on FunctionName. In Windows, you can use the following syntax to pipe the output of list-functions into the findstr command and redirect the result into a text file.

aws lambda list-functions | findstr /C:FunctionName >lambdafns.txt
The resulting file looks like this:

```
"FunctionName": "CloudGemSamples-CGSamplesDep-Clo-ServiceLambda-hash",
"FunctionName": "CloudGemSamples-AH-CoreResourceTypes-AWS_SQS_Queue",
"FunctionName": "CloudGemSamples-AH-CoreResourceTypes-Custom_ServiceApi",
"FunctionName": "CloudGemSamples-CRH-CoreResourceTypes-Custom_Helper",
"FunctionName": "CloudGemSamples-CRH-CoreResourceTypes-Custom_Interface-hash",
"FunctionName": "CloudGemSamples-CGSamplesDep-PackageVoiceLines-hash",
```

As before, use search and replace operations to turn the file into a deletion script, as in the following example.

```
aws lambda delete-function --function-name CloudGemSamples-CGSamplesDep13-Clo-ServiceLambda-hash
aws lambda delete-function --function-name CloudGemSamples-AH-CoreResourceTypes-AWS_SQS_Queue
aws lambda delete-function --function-name CloudGemSamples-AH-CoreResourceTypes-Custom_ServiceApi
aws lambda delete-function --function-name CloudGemSamples-CRH-CoreResourceTypes-Custom_Helper
aws lambda delete-function --function-name CloudGemSamples-CRH-CoreResourceTypes-Custom_Interface-hash
aws lambda delete-function --function-name CloudGemSamples-CGSamplesDep13-PackageVoiceLines-hash
```

For more information on the AWS Lambda CLI commands, see `lambda`

**Amazon API Gateway**

To delete a REST API from API Gateway, you must specify the ID of the REST API.

```
aws apigateway delete-rest-api --rest-api-id rest_api_id
```

To list the APIs in API Gateway, use the `get-rest-apis` command.

```
aws apigateway get-rest-apis
```

In Windows, you can filter the output of the `get-rest-apis` command by ID as in the following example.

```
aws apigateway get-rest-apis | findstr /C:id >gatewayapis.txt
```

For more information on the API Gateway CLI commands, see `apigateway`

**Amazon S3**

In Amazon S3, you can **delete** empty buckets but must **remove** non-empty buckets.

**Note**
The commands presented here work only on unversioned buckets. Amazon S3 buckets are unversioned by default. For more information, see **Using Versioning** in the **Amazon Simple Storage Service User Guide**.

**Getting a List of Bucket Names**

To obtain a list of S3 bucket names, use the following command.
aws s3api list-buckets --query Buckets[].Name

Deleting a Bucket

The following syntax deletes an empty S3 bucket:

aws s3api delete-bucket --bucket bucket_name

Removing a Bucket

To remove a non-empty bucket, use the `rb` (remove bucket) command with `--force` parameter.

aws s3 rb s3://bucket_name --force

The following example command deletes all objects in the `cloudgemsamples-cloudgemportal-hash` bucket and then deletes the bucket itself.

aws s3 rb s3://cloudgemsamples-cloudgemportal-hash --force

For more information on the Amazon S3 CLI high-level commands, see s3. For more information on the more detailed Amazon S3 CLI commands, see s3api.

Using the AWS Management Console

You can use the AWS Management Console to manually delete individual AWS resources from your account. However, if you have many resources to delete, the Cloud Canvas cleanup tool or the AWS CLI are faster alternatives.

To delete an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Identify a bucket that you no longer require.
3. Select the identified bucket’s line entry.
4. Click the Empty Bucket button.
5. Follow modal instructions to empty bucket and confirm.
6. Click the Delete Bucket button.
7. Follow the modal instructions to delete the bucket and confirm.

To delete an API from API Gateway

1. Sign in to the AWS Management Console and open the API Gateway console at https://console.aws.amazon.com/apigateway/.
2. In the upper-right corner of the management console, choose the AWS region from which you want to delete resources.
3. Identify an API that you no longer require.
4. Click the API.
5. Click the Actions menu.
6. Select Delete API from the menu.
7. Follow the instructions in the DeleteRestApi dialog box to confirm the deletion of the API.

To delete AWS CloudFormation stacks

2. In the upper-right corner of the management console, choose the AWS region from which you want to delete resources.
3. Identify a AWS CloudFormation stack that you no longer require.
4. Select the stack.
5. Click the Actions menu.
6. Choose Delete Stack from the drop-down list.

   **Note**
   If termination protection is enabled for the stack, you must remove the protection before you can delete the stack.
7. Follow the instructions to confirm the deletion of the stack.

Understanding the Resource Manager Security System

Cloud Canvas Resource Manager provides a robust set of features for securing your Amazon Lumberyard project and its cloud-connected resources. You can control the access to project resources of team members, players, Lambda functions, and specify other custom access control configuration for your project.

**Topics**

- Controlling Access to Resources (p. 2370)
- Cloud Canvas Built-In Roles and Policies (p. 2373)
- Player Identity (p. 2379)
- Using the Cloud Canvas Command Line to Manage Roles and Permissions (p. 2381)
- Improve Security with a Custom Domain Name (p. 2386)

Controlling Access to Resources

Setting access permissions correctly is key to ensuring that Cloud Canvas Resource Manager manages your project's cloud-connected features securely.

Access Scenarios and ProjectResourceHandler

Cloud Canvas Resource Manager requires that the following access scenarios be supported. Additional roles with more nuanced permissions can be created, but the table below describes the core access requirements.
A project team member must be able to create resource group stacks that contain arbitrary resources, but not be able to create or modify roles and policies. This introduces a significant complexity. Some resources like Lambda functions require that the game developer also provide a role that is assumed by the resource. The game developer must be able to create such roles and manage their policies. However, granting IAM permissions such as these directly to team members would effectively make them administrators.

To enable the required functionality while still limiting what a project team member can do directly, Cloud Canvas Resource Manager uses AWS CloudFormation custom resources. The custom resource handlers for Cloud Canvas Resource Manager are implemented in the `ProjectResourceHandler` Lambda function in the project stack. The Lambda function's execution role (`ProjectResourceHandlerExecution`) grants permissions that Cloud Canvas Resource Manager requires. These permissions are not granted to project team members.

For example, the `Custom::AccessControl` resource, described in detail later in this document, is responsible for managing inline policies on various roles. It can perform these actions on the project team member's behalf. However, the `Custom::AccessControl` handler also must know what to put into these policies. It can't trust the project team member to provide this information directly. Instead, it must construct the information from trusted sources. To do this, `AccessControl` uses metadata on resource definitions from AWS CloudFormation. It also constructs ARNs for the stack's resources identified by AWS CloudFormation. In this way, only a user with permission to update the stack can influence the policies that are constructed for the resources in that stack.

<table>
<thead>
<tr>
<th>Project Stack</th>
<th>Project Team Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Access</td>
<td>Read</td>
</tr>
<tr>
<td>Full Access</td>
<td>Read/Update</td>
</tr>
<tr>
<td>Full Access</td>
<td>Read</td>
</tr>
<tr>
<td>Full Access</td>
<td>Create/Update/Delete</td>
</tr>
<tr>
<td>Full Access</td>
<td>Assume</td>
</tr>
</tbody>
</table>

A project team member must be able to create resource group stacks that contain arbitrary resources, but not be able to create or modify roles and policies. This introduces a significant complexity. Some resources like Lambda functions require that the game developer also provide a role that is assumed by the resource. The game developer must be able to create such roles and manage their policies. However, granting IAM permissions such as these directly to team members would effectively make them administrators.

To enable the required functionality while still limiting what a project team member can do directly, Cloud Canvas Resource Manager uses AWS CloudFormation custom resources. The custom resource handlers for Cloud Canvas Resource Manager are implemented in the `ProjectResourceHandler` Lambda function in the project stack. The Lambda function's execution role (`ProjectResourceHandlerExecution`) grants permissions that Cloud Canvas Resource Manager requires. These permissions are not granted to project team members.

For example, the `Custom::AccessControl` resource, described in detail later in this document, is responsible for managing inline policies on various roles. It can perform these actions on the project team member's behalf. However, the `Custom::AccessControl` handler also must know what to put into these policies. It can't trust the project team member to provide this information directly. Instead, it must construct the information from trusted sources. To do this, `AccessControl` uses metadata on resource definitions from AWS CloudFormation. It also constructs ARNs for the stack's resources identified by AWS CloudFormation. In this way, only a user with permission to update the stack can influence the policies that are constructed for the resources in that stack.
Using the Custom::AccessControl Resource

As described above, Cloud Canvas Resource Manager security depends on IAM roles and the credentials used from assuming such roles. The Access Scenarios and ProjectResourceHandler (p. 2370) section earlier in this topic explains why Cloud Canvas Resource Manager has the responsibility of managing the inline policies attached to these roles.

This section describes the data used by the Custom::AccessControl resource handler to configure the project's roles. A Custom::AccessControl resource must be defined in the following templates:

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project-template.json</td>
<td>Causes policies on the roles defined in the project-template.json file to be updated. These roles can provide access to any resource defined in any resource groups across all deployments.</td>
</tr>
<tr>
<td>deployment-access-template.json</td>
<td>Causes policies on the roles defined in the deployment-access-template.json file to be updated. These roles can provide access to any resource in any resource of a given deployment.</td>
</tr>
<tr>
<td>resource-group-template.json</td>
<td>Causes policies on the roles defined in the project-template.json and deployment-access-template.json files to be updated. Only permissions for the resource defined in the resource-group-template.json file are updated. For roles defined in the deployment-access-template.json file, only the instances of those roles for the deployment that contains the resource group stack are updated.</td>
</tr>
</tbody>
</table>

This process is illustrated in the following diagram. The diagram shows the metadata that is read and roles that are updated when a resource group stack, deployment access stack, or project stack is updated.
Custom::AccessControl Resource Definitions

The Custom::AccessControl resource supports the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationBucket</td>
<td>The name of the project’s configuration bucket. This property must be provided.</td>
</tr>
<tr>
<td>ConfigurationKey</td>
<td>Identifies the location in the configuration bucket where data for the stack operation is stored. However, the custom resource handler depends on this value changing on each update. Property changes such as this cause AWS CloudFormation to invoke the custom resource handler on each stack operation.</td>
</tr>
<tr>
<td>ServiceToken</td>
<td>Identifies the Lambda function that is invoked for the custom resource. This should be the project global ProjectResourceHandler Lambda function that is defined in the project-template.json file.</td>
</tr>
</tbody>
</table>

The DependsOn attribute of the Custom::AccessControl resource definition must list the following resources.

- All the resources in the project-template.json, deployment-access-template.json, or resource-group-template.json files that provide permissions metadata.

- All the AWS::IAM::Role resources that have RoleMapping metadata.

- Any custom resources that create implicit roles, such as Custom::LambdaConfiguration and Custom::ServiceApi resources.

When you use the AWS CLI to manage roles and permissions, these resources are listed for you. However, if you edit these files yourself, it is important that you maintain these dependencies. Without these dependencies, the Custom::AccessControl resource might be updated before the other resources have been updated. If this occurs, the Custom::AccessControl no longer has access to the latest metadata from the resources, and the changes that were intended might not be made.

Setting Access Permissions

For information on setting Custom::AccessControl permissions, see Permissions Metadata for Resource Definitions (p. 2240).

Cloud Canvas Built-In Roles and Policies

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the built-in Cloud Canvas roles and policies to manage resource and deployment permissions for your project.
**Built-In Roles**

You can use the `AWS::IAM:Role` resource to define roles in your `project-template.json` or `deployment-access-template.json` files. Cloud Canvas Resource Manager defines the following roles for you.

**DeploymentAdmin**

Grants restricted access to the deployment. Similar to the `DeploymentOwner` role, but cannot create or delete resource group stacks. This is a convenience role that is safer than `DeploymentOwner`. To add additional restrictions to the `DeploymentAdmin` role, edit the `DeploymentAdminRestrictions` policy definition.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by principles defined in the same AWS account, as allowed by their permissions.</td>
</tr>
<tr>
<td>Attached</td>
<td><code>DeploymentAccess, DeploymentOwnerAccess, and DeploymentAdminRestrictions</code></td>
</tr>
<tr>
<td>Inline</td>
<td>Added and removed by the <code>Custom::AccessControl</code> resource handler based on abstract role mappings.</td>
</tr>
</tbody>
</table>

File location: `\project_name\deployment_name\deployment-access-template.json`.

**DeploymentOwner**

Grants full access to all the resources in a deployment. To modify the default permissions granted by this role, edit the `DeploymentOwnerAccess` policy definition.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by principles defined in the same AWS account, as allowed by their permissions.</td>
</tr>
<tr>
<td>Attached</td>
<td><code>DeploymentAccess and DeploymentOwnerAccess</code></td>
</tr>
<tr>
<td>Inline</td>
<td>Added and removed by the <code>Custom::AccessControl</code> resource handler based on abstract role mappings.</td>
</tr>
</tbody>
</table>

File location: `\project_name\deployment_name\deployment-access-template.json`.

**Note**

Permissions added to `DeploymentOwnerAccess` are also granted to `DeploymentAdmin` unless they are denied by `DeploymentAdminRestrictions`.

**Player**

Grants players limited access to specific resources in a deployment as determined by permissions metadata on those resources.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by <code>cognito-identity.amazonaws.com</code>.</td>
</tr>
<tr>
<td>Attached</td>
<td>None.</td>
</tr>
</tbody>
</table>
## Administering Cloud Canvas

### Policy Type

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline</td>
<td>Added and removed by the <code>Custom::AccessControl</code> resource handler based on abstract role mappings.</td>
</tr>
</tbody>
</table>

File location: `\project_name\deployment_name\deployment-access-template.json`.

### ProjectAdmin

Grants project administrators restricted access to all the project's deployments. Similar to the `ProjectOwner` role, but cannot create, update, or delete deployments with names that start with "Release." This is a convenience role that is safer than `ProjectOwner`. To add additional restrictions to the `ProjectAdmin` role, edit the `ProjectAdminRestrictions` definition.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by principals defined in the same AWS account, as allowed by their permissions.</td>
</tr>
<tr>
<td>Attached</td>
<td><code>ProjectAccess</code>, <code>ProjectOwnerAccess</code>, and <code>ProjectAdminRestrictions</code>.</td>
</tr>
<tr>
<td>Inline</td>
<td>Added and removed by the <code>Custom::AccessControl</code> resource handler based on abstract role mappings.</td>
</tr>
</tbody>
</table>

File location: `\project_name\project-template.json`.

**Warning**
The intent of this role is to securely "sandbox" user actions so that they don't accidentally impact other projects. However, anyone who can assume the `ProjectAdmin` role can grant themselves additional permissions. Because a `ProjectAdmin` user can escalate the privilege for the role, the `ProjectAdmin` role should still be considered an account administrator role and therefore a potential security concern.

### ProjectOwner

Grants project administrators full access to all the project's resources. To modify the default permissions granted by this role, edit the `ProjectOwnerAccess` policy definition.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by principals defined in the same AWS account, as allowed by their permissions.</td>
</tr>
<tr>
<td>Attached</td>
<td><code>ProjectAccess</code> and <code>ProjectOwnerAccess</code>.</td>
</tr>
<tr>
<td>Inline</td>
<td>Added and removed by the <code>Custom::AccessControl</code> resource handler based on abstract role mappings.</td>
</tr>
</tbody>
</table>

File location: `\project_name\project-template.json`.

**Warning**
- The intent of this role is to securely "sandbox" user actions so that they don't accidentally impact other projects. However, anyone who can assume the `ProjectOwner` role can grant themselves additional permissions. Because a `ProjectOwner` user can escalate the privilege for the role, the `ProjectOwner` role should still be considered an account administrator role and therefore a potential security concern.
for the role, the ProjectOwner role should still be considered an account administrator role and therefore a potential security concern.

- Permissions added to ProjectOwnerAccess are also granted to ProjectAdmin unless they are denied by ProjectAdminRestrictions.

**ProjectResourceHandlerExecution**

Grants the ProjectResourceHandler Lambda function runtime execution permissions. This role grants permissions for Cloud Canvas Resource Manager to use AWS CloudFormation custom resources for stack operations. For more information, see Access Scenarios and the ProjectResourceHandler (p. 2370).

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by the AWS Lambda service.</td>
</tr>
<tr>
<td>Attached</td>
<td>None.</td>
</tr>
<tr>
<td>Inline</td>
<td>ProjectAccess.</td>
</tr>
</tbody>
</table>

File location: `\project_name\project-template.json`.

**Server**

Grants Lumberyard dedicated server builds access to select deployment resources, as determined by permissions metadata on those resources.

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume role</td>
<td>Can be assumed by cognito-identity.amazonaws.com.</td>
</tr>
<tr>
<td>Attached</td>
<td>None.</td>
</tr>
<tr>
<td>Inline</td>
<td>Added and removed by the Custom::AccessControl resource handler based on abstract role mappings.</td>
</tr>
</tbody>
</table>

File location: `\project_name\deployment_name\deployment-access-template.json`.

**Role Scope**

The configuration file in which you define a role determines the resources to which the role provides access.

<table>
<thead>
<tr>
<th>File</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>project-template.json</td>
<td>Applies to all resources in all resource groups for all deployments. Only a single instance of the role is created for the entire project.</td>
</tr>
<tr>
<td>deployment-template.json</td>
<td>Applies to all resources in all resource groups for a specific deployment. A separate instance of the role is created for each deployment.</td>
</tr>
</tbody>
</table>

You can use the `lmbr_aws` command line tool to manage the role definitions in the `project-template.json` and `deployment-access-template.json` files. For more information, see Using the Cloud Canvas Command Line to Manage Roles and Permissions (p. 2381).
**Implicit Roles**

Some Cloud Canvas custom resources also create roles. For example, when a Lambda function is executed, it assumes the role that the `Custom::LambdaConfiguration` resource creates. When API Gateway invokes a Lambda function or accesses other resources, it assumes the role that the `Custom::ServiceApi` resource creates. Including these custom resources in a `resource-group-template.json` file causes these implicit roles to be created (and deleted when the resource is deleted). For information on implicit role names, see Implicit Role Mappings (p. 2379).

**Managed Policies**

You can use `AWS::IAM::ManagedPolicy` resources to define permissions that are shared across any number of roles. Cloud Canvas defines the following managed policies for you:

<table>
<thead>
<tr>
<th>Policy</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectAccess</td>
<td><code>project-template.json</code></td>
<td>Defines the permissions needed to access the project and deployment configuration that must be read before a project-wide role can be assumed. For more information, see Assuming a Role (p. 2381).</td>
</tr>
<tr>
<td>ProjectOwnerAccess</td>
<td><code>project-template.json</code></td>
<td>Defines the default permissions granted to the ProjectOwner and ProjectAdmin roles.</td>
</tr>
<tr>
<td>ProjectAdminRestrictions</td>
<td><code>project-template.json</code></td>
<td>Defines restrictions to the ProjectOwnerAccess policy that apply only to the ProjectAdmin role.</td>
</tr>
<tr>
<td>DeploymentAccess</td>
<td><code>deployment-access-template.json</code></td>
<td>Defines the permissions needed to access project and deployment configuration that must be read before a deployment-specific role can be assumed. For more information, see Assuming a Role (p. 2381).</td>
</tr>
<tr>
<td>DeploymentOwnerAccess</td>
<td><code>deployment-access-template.json</code></td>
<td>Defines the default permissions granted to the DeploymentOwner and DeploymentAdmin roles.</td>
</tr>
<tr>
<td>DeploymentAdminRestrictions</td>
<td><code>deployment-access-template.json</code></td>
<td>Defines restrictions to the DeploymentOwnerAccess policy that apply only to the DeploymentAdmin role.</td>
</tr>
</tbody>
</table>

The `ProjectAdmin` and `DeploymentAdmin` roles are granted the same permissions as the `ProjectOwner` and `DeploymentOwner` roles, minus any permissions specifically denied by the `ProjectAdminRestrictions` and `DeploymentAdminRestrictions` managed policies, respectively. In effect, an "admin" is granted all the permissions of an "owner" minus any special actions that the "admin" should not be able to perform.

**Role Mapping Metadata**

The `AbstractRole` property in the `Permission` metadata object does not directly specify the actual role that receives the described permission. These values must be mapped to actual IAM roles. This makes it possible to setup roles in whatever way makes sense for your project. It also removes the need to modify the permissions defined by individual resource groups.

The ability to map abstract roles to actual IAM roles is important when you use a cloud gem across multiple projects or from a third party. Cloud gems acquired from a third party might have roles that are different from the roles that you use in your organization. (A cloud gem is a Lumberyard gem (p. 1064) that uses the AWS resources defined by a Cloud Canvas Resource Group. For more information, see Cloud Gems (p. 2127).)
The Custom::AccessControl resource looks for CloudCanvas RoleMappings metadata on AWS::IAM::Role resources to determine which abstract roles map to that physical role. In the following example, the CustomerSupport abstract role from all resource groups is mapped to the DevOps physical role.

```
... 

"DevOps": { 
  "Type": "AWS::IAM::Role",
  "Properties": { 
    "Path": "Fn::Join": [ "", [ "/", { "Ref": "ProjectStack" }, "/", { "Ref": "DeploymentName" } ] ]
  },
  "Metadata": { 
    "CloudCanvas": { 
      "RoleMappings": [ 
        { 
          "AbstractRole": [ "*.CustomerSupport" ],
          "Effect": "Allow"
        }
      ]
    }
  },
},
... 
```

Each Cloud Canvas RoleMapping metadata object can have the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractRole</td>
<td>Required string or list of strings of the form <code>&lt;resource-group-name&gt;.&lt;abstract-role-name&gt;</code>. To match abstract roles defined in any resource group, use * for the <code>&lt;resource-group-name&gt;</code>. To match abstract roles defined only in a specific resource group, use an actual resource group name.</td>
</tr>
<tr>
<td>Effect</td>
<td>Required string. Must be either <code>Allow</code>, to allow an action defined by permission metadata, or <code>Deny</code> to deny an action. This is used as the Effect property in a statement in the policy.</td>
</tr>
</tbody>
</table>

You can use the `lmbr_aws` command line tool to manage RoleMappings metadata on role resource definitions in the `project-template.json` and `deployment-access-template.json` files. For more information, see Using the Cloud Canvas Command Line to Manage Roles and Permissions.

**Default Role Mappings**

Cloud Canvas defines role mappings for the following roles:

<table>
<thead>
<tr>
<th>Role</th>
<th>File</th>
<th>Default Role Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectResourceHandlerExecution</td>
<td>project-template.json</td>
<td>none</td>
</tr>
<tr>
<td>ProjectOwner</td>
<td>project-template.json</td>
<td>*.ProjectAdmin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.ProjectOwner</td>
</tr>
<tr>
<td>ProjectAdmin</td>
<td>project-template.json</td>
<td>*.ProjectAdmin</td>
</tr>
<tr>
<td>DeploymentOwner</td>
<td>deployment-access-template.json</td>
<td>*.DeploymentAdmin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*.DeploymentOwner</td>
</tr>
</tbody>
</table>
Implicit Role Mappings

As mentioned in Implicit Roles (p. 2377), role mappings are automatically defined for the implicit roles created by Cloud Canvas resources like Custom::LambdaConfiguration. These mappings are only used with permission metadata in the same resource-group-template.json file as the custom resource that creates the role. The name of the abstract role used in permission metadata to reference an implicit role depends on the custom resource type.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Implicit Role Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom::LambdaConfiguration</td>
<td>The name of the Lambda function, as specified by the Function property in the Custom::LambdaConfiguration resource definition. This is also the logical ID of the AWS::Lambda::Function resource.</td>
</tr>
<tr>
<td>Custom::ServiceApi</td>
<td>The logical ID of the Custom::ServiceApi resource.</td>
</tr>
</tbody>
</table>

Player Identity

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the AWS Game Tech blog to learn more.

A Lumberyard cloud-connected game must use AWS credentials that grant the desired access when calling AWS APIs (using either the C++ AWS SDK or the AWS flow nodes). Cloud Canvas uses an Amazon Cognito identity pool to get these credentials.

Using a Amazon Cognito identity pool has the benefit of providing the game with a unique identity for each individual player. This identity can be used to associate the player with their saved games, high scores, or any other data stored in DynamoDB tables, Amazon S3 buckets, or other locations. If you use the Player Account Cloud Gem, this identity is mapped to an account ID that can be used instead.

Amazon Cognito identity pools support both unauthenticated and authenticated identities. Unauthenticated identities are associated with a single device such as a PC, tablet, or phone, and have no associated user name or password.

Authenticated identities are associated with the identity of an player as determined by an external identity provider such as Amazon, Facebook, or Google, or an Amazon Cognito user pool. This allows Amazon Cognito to provide the game with the same player identity everywhere a player plays a game. The player’s saved games, high scores, and other data effectively follow the player from device to device.

Cloud Canvas supports both anonymous (unauthenticated) and authenticated player identities. The Player Account Cloud Gem is the easiest way to add authentication. Authenticated identity support for other providers is more complex and requires additional setup and coding.
Anonymous (Unauthenticated) Player Login

In Lumberyard version 1.11, the identity system that previously existed in an LmbrAWS CryEngine module (`dev\Code\CryEngine\CryCommon\LmbrAWS`) has been converted into a CloudCanvasPlayerIdentityComponent (`dev\Gems\CloudGemFramework\vN\Code\Source\PlayerIdentity.*`). This component is implemented as a required system component from the CloudGemFramework Gem. The Cloud Canvas client configuration system is automatically initialized by the following call:

```cpp
EBUS_EVENT_RESULT(appliedConfiguration, CloudGemFramework::CloudCanvasPlayerIdentityBus, ApplyConfiguration)
```

Alternatively, you can use a Cloud Canvas (AWS):Configuration:ApplyConfiguration flow node. An existing anonymous identity is loaded from the local identities cache. If an anonymous identity is not found in the cache, a new identity is acquired from the pool. The identity cache is stored in an .aws/.identities file in the user's home directory.

Authenticated Player Login

In order to understand how to use Cloud Canvas to implement authenticated player identities for your game, you must be familiar with Amazon Cognito's Enhanced (Simplified) Authflow. For information, see the article Authentication Flow in the Amazon Cognito Developer Guide.

The login process for authenticated player identities is more complex than the anonymous player login process. The Player Account Cloud Gem handles this for Amazon Cognito user pools. For other providers, this login process requires additional setup beyond what Cloud Canvas provides by default.

The authenticated player login process takes place automatically when the Cloud Canvas client configuration system is initialized by the following call:

```cpp
EBUS_EVENT_RESULT(appliedConfiguration, CloudGemFramework::CloudCanvasPlayerIdentityBus, ApplyConfiguration)
```

Alternatively, the process can occur by using a Cloud Canvas (AWS):Configuration:ApplyConfiguration flow node. If an authenticated identity is found in the local identities cache, it is loaded and the access token is refreshed if needed using the stored refresh token. If no existing identity is found, it falls back to using the anonymous identity for AWS calls.

The code that implements the authenticated login flow can be found in the `dev\Gems\CloudGemFramework\vN\Code\Source\Identity` directory. A description of the files follows.

- **ResourceManagementLambdaBasedTokenRetrievalStrategy.cpp** – Implements the token exchange process that calls the PlayerAccessTokenExchange Lambda function.

- **TokenRetrievingPersistentIdentityProvider.cpp** – An implementation of the PersistentCognitoIdentityProvider interface that is defined in the AWS SDK. The implementation uses ResourceManagementLambdaBasedTokenRetrievalStrategy instances to implement the token exchange process.

Configuring a Cognito Identity Provider (Cognito User Pool)

Cloud Canvas provides the `Custom::CognitoUserPool (p. 2253)` resource for adding the Amazon Cognito user pools and linking them to a Amazon Cognito identity pool. The Player Account Cloud Gem (p. 2188) uses this custom resource. It also provides an EBus interface so that you can work with the user pool and the sample level that has an in-game menu.
You can use the `lmbr_aws` command line tool to manage Cloud Canvas Resource Manager access control. For example, you can use the tool to assume a role when you run a command, or to manage roles, permissions, and role mappings.

### Assuming a Role

Most `lmbr_aws` commands support an `--assume-role <role-name>` argument. You can use this argument to assume a role when you run a command.

If specified, `<role-name>` must be the logical resource ID of an IAM role resource defined in either the `project-template.json` or `deployment-access-template.json` files.

**Note**

You should avoid defining roles that have the same name in both files. If you do, the role from the deployment file takes precedence.

If you specify a deployment access role, the actual role used depends on the deployment on which the command is operating. If the `--deployment` argument has been specified, then the specified deployment is used. If the `--deployment` argument has not been specified and the user has specified a default deployment, the default deployment is used. If a default deployment has not been specified, the project's default deployment is used.

`lmbr_aws` uses your configured AWS credentials to assume the specified role. See [Configuration](#) for a description of how the credentials are determined. The credentials must have permission to assume the role. For more information, see [Granting a User Permission to Switch Roles](#).

Before assuming the role, the `lmbr_aws` tool uses credentials to read project configuration data from AWS. The `ProjectAccess` managed policy in the `project-template.json` file and the `DeploymentAccess` managed policy in the `deployment-access-template.json` file grant the permissions necessary to read this information. You can attach the corresponding managed policy to any IAM user that works on a project or deployment.

Note that administrative users created for an AWS account normally have permissions to assume roles and read project configuration. Administrative users typically have permission to perform any action on any resource owned by an account.

### Role Management Commands

Role management commands manage the `AWS::IAM::Role` resource definitions in the `project-template.json` and `deployment-access-template.json` files. After you use these commands to make changes, you must update the project or deployment access stacks for the changes to take effect. For information about the permissions to perform this action, see [Controlling Access to Resources](#).

#### `lmbr_aws role add`

Adds an AWS IAM role resource definition to the `project-template.json` file or `deployment-access-template.json` file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>role</code></td>
<td>Required. The name of the role resource definition.</td>
</tr>
</tbody>
</table>
**lmbr.aws role remove**

Removes an AWS IAM role resource definition from the `project-template.json` file or `deployment-access-template.json` file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>##role <code>&lt;role-name&gt;</code></td>
<td>Required. The name of the role resource definition.</td>
</tr>
<tr>
<td>##project</td>
<td>Optional. When present, specifies that the role resource definition be removed from the <code>project-template.json</code> file. Otherwise, the role resource definition is removed from the <code>deployment-access-template.json</code> file.</td>
</tr>
</tbody>
</table>

**lmbr.aws role list**

Lists the AWS IAM role definitions in the `project-template.json` and/or `deployment-access-template.json` files.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>##deployment</td>
<td>Optional. Either --deployment or --project can be specified. If --deployment is specified, only the roles in the <code>deployment-access-template.json</code> file are listed.</td>
</tr>
<tr>
<td>##project</td>
<td>Optional. Either --deployment or --project can be specified. If --project is specified, only the roles in the <code>project-template.json</code> file are listed.</td>
</tr>
</tbody>
</table>

**Output**

The output is similar to the following example.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>DeploymentAdmin</td>
</tr>
<tr>
<td>Deployment</td>
<td>DeploymentOwner</td>
</tr>
<tr>
<td>Deployment</td>
<td>Player</td>
</tr>
<tr>
<td>Deployment</td>
<td>PlayerLoginRole</td>
</tr>
<tr>
<td>Project</td>
<td>ProjectAdmin</td>
</tr>
<tr>
<td>Project</td>
<td>ProjectOwner</td>
</tr>
<tr>
<td>Project</td>
<td>PlayerAccessTokenExchangeExecution</td>
</tr>
<tr>
<td>Project</td>
<td>ProjectResourceHandlerExecution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Indicates whether the role is defined in the <code>deployment-access-template.json</code> or <code>project-template.json</code> file.</td>
</tr>
<tr>
<td>Name</td>
<td>Shows the resource definition name. This is the &quot;logical&quot; resource name, not the &quot;physical&quot; resource name which identifies an actual instance of the role. To see the physical resource</td>
</tr>
</tbody>
</table>
Permission Metadata Management

The permission metadata management commands manage CloudCanvas Permissions metadata on resource definitions in the resource-group-template.json files. After you use these commands to make changes, you must update the project or deployment access stacks for the changes to take effect. For information about the permissions to perform this action, see Controlling Access to Resources.

lmbr_aws permission add

Adds Cloud Canvas Permissions metadata to an resource definition in a resource-group-template.json file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>##resource#group &lt;resource-group-name&gt;</td>
<td>Required. The name of a resource group. The metadata will be added to a resource definition in that resource group's resource-group-template.json file.</td>
</tr>
<tr>
<td>##resource &lt;resource-name&gt;</td>
<td>Required. The name of the resource definition in the resource-group-template.json file.</td>
</tr>
<tr>
<td>##role &lt;abstract-role-name&gt;</td>
<td>Required. Identifies the role that is granted the permission.</td>
</tr>
<tr>
<td>##action &lt;action&gt; [&lt;action&gt; ...]</td>
<td>Required. The action that is allowed. You can specify more than one action.</td>
</tr>
<tr>
<td>##suffix &lt;suffix&gt; [&lt;suffix&gt; ...]</td>
<td>Optional. A string appended to the resource ARN. You can specify more than one suffix.</td>
</tr>
</tbody>
</table>

lmbr_aws permission remove

Removes Cloud Canvas Permissions metadata from a resource definition in a resource-group-template.json file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>##resource#group &lt;resource-group-name&gt;</td>
<td>Required. The name of a resource group. The metadata is removed from a resource definition in the specified resource group's resource-group-template.json file.</td>
</tr>
<tr>
<td>##resource &lt;resource-name&gt;</td>
<td>Required. The name of the resource definition in the resource-group-template.json file.</td>
</tr>
<tr>
<td>##role &lt;abstract-role-name&gt;</td>
<td>Required. Identifies the roles from which permissions are removed.</td>
</tr>
<tr>
<td>##action &lt;action&gt;</td>
<td>Optional. The action that is removed. You can specify more than one action. If not specified, all permissions for the role are removed.</td>
</tr>
<tr>
<td>##suffix &lt;suffix&gt;</td>
<td>Optional. A string appended to the resource ARN, which is removed. You can specify more than one suffix.</td>
</tr>
</tbody>
</table>
Imbr/aws permission list

Removes Cloud Canvas Permissions metadata from an resource definition in a resource-group-template.json file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>###resource#group &lt;resource-name&gt;</td>
<td>Optional. The name of the resource definition in the resource-group-template.json file. The default lists metadata from all resource definitions.</td>
</tr>
<tr>
<td>###role &lt;abstract-role-name&gt;</td>
<td>Optional. Lists metadata for the specified abstract role. The default lists metadata for all abstract roles.</td>
</tr>
</tbody>
</table>

Output

The output is similar to the following example.

<table>
<thead>
<tr>
<th>Resource Group</th>
<th>Resource</th>
<th>Resource Type</th>
<th>Roles</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynamicContent</td>
<td>ContentBucket</td>
<td>AWS::S3::Bucket</td>
<td>ServiceLambda</td>
<td>s3:GetObject</td>
</tr>
<tr>
<td>DynamicContent</td>
<td>ContentBucket</td>
<td>AWS::S3::Bucket</td>
<td>ContentRequest</td>
<td>s3:*</td>
</tr>
<tr>
<td>DynamicContent</td>
<td>ContentRequest</td>
<td>AWS::Lambda::Function</td>
<td>Player</td>
<td></td>
</tr>
<tr>
<td>DynamicContent</td>
<td>ServiceLambda</td>
<td>AWS::Lambda::Function</td>
<td>ServiceApi</td>
<td></td>
</tr>
<tr>
<td>DynamicContent</td>
<td>StagingSettingsTable</td>
<td>AWS::DynamoDB::Table</td>
<td>ServiceLambda</td>
<td>dynamodb:GetItem, dynamodb:Scan, dynamodb:UpdateItem</td>
</tr>
<tr>
<td>DynamicContent</td>
<td>StagingSettingsTable</td>
<td>AWS::DynamoDB::Table</td>
<td>ContentRequest</td>
<td>dynamodb:GetItem</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Group</td>
<td>Shows the resource group where the permission metadata was found.</td>
</tr>
<tr>
<td>Resource</td>
<td>Shows the name of the resource definition with the metadata.</td>
</tr>
<tr>
<td>Resource Type</td>
<td>Shows the type of the resource definition with the metadata.</td>
</tr>
<tr>
<td>Roles</td>
<td>Shows the abstract roles specified by the permission metadata.</td>
</tr>
<tr>
<td>Actions</td>
<td>Shows the actions specified by the permission metadata.</td>
</tr>
<tr>
<td>ARN Suffixes</td>
<td>Shows the suffix added to the resource ARN, as specified by the permission metadata.</td>
</tr>
</tbody>
</table>

Tip

To see all the resources players have access to through the game client, use the command:
Role Mapping Metadata Management

Role mapping metadata management commands manage CloudCanvas RoleMappings metadata on AWS::IAM::Role resource definitions in the project-template.json and deployment-access-template.json files. After you use these commands to make changes, you must update the project or deployment access stacks for the changes to take effect. For information about the permissions to perform this action, see Controlling Access to Resources (p. 2370).

**lmbr_aws role-mapping add**

Adds Cloud Canvas RoleMappings metadata to an AWS IAM role definition in the project-template.json or deployment-access-template.json file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>##role &lt;role-name&gt;</td>
<td>Required. The name of the role resource definition.</td>
</tr>
</tbody>
</table>
| ##pattern <abstract-role-pattern> | Identifies the abstract roles mapped to the role. Has the form <resource-group-name>.<abstract-role-name>, where <resource-group-name> can be *.
| ##allow       | Either --allow or --deny must be specified. Indicates that the permissions requested for the abstract role are allowed. |
| ##deny        | Either --allow or --deny must be specified. Indicates that the permissions requested for the abstract role are denied. |
| ##project     | Optional. Indicates that the role definition is in the project-template.json file. The default is for the role definition to be in the deployment-access-template.json file. |

**lmbr_aws role-mapping remove**

Removes an AWS IAM role resource definition from the project-template.json file or deployment-access-template.json file.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>##role &lt;role-name&gt;</td>
<td>Required. The name of the role resource definition.</td>
</tr>
</tbody>
</table>
| ##pattern <abstract-role-pattern> | Identifies the abstract roles mapped to the role. Has the form <resource-group-name>.<abstract-role-name>, where <resource-group-name> can be *.
| ##project     | Optional. Indicates that the role definition is in the project-template.json file. The default is for the role definition to be in the deployment-access-template.json file. |

**lmbr_aws role-mapping list**

Lists the AWS IAM role definitions in the project-template.json and/or deployment-access-template.json files.
## Administering Cloud Canvas

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>##role &lt;role-name&gt;</code></td>
<td>Required. The role definition with the metadata to list. The default is to list metadata from all role definitions.</td>
</tr>
<tr>
<td><code>##pattern &lt;abstract-role-pattern&gt;</code></td>
<td>The abstract role pattern specified by the metadata listed. The default is to list metadata with any abstract role pattern.</td>
</tr>
<tr>
<td><code>##deployment</code></td>
<td>Optional. Either <code>--deployment</code> or <code>--project</code> can be specified. Lists metadata from role definitions in the <code>deployment-access-template.json</code> file. The default is to list metadata from role definitions in the <code>project-template.json</code> file and <code>deployment-access-template.json</code> files.</td>
</tr>
<tr>
<td><code>##project</code></td>
<td>Optional. Either <code>--deployment</code> or <code>--project</code> can be specified. Lists metadata from role definitions in the <code>project-template.json</code> file. The default is to list metadata from role definitions in the <code>project-template.json</code> file and <code>deployment-access-template.json</code> files.</td>
</tr>
</tbody>
</table>

**Output**

The output is similar to the following example.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Actual Role</th>
<th>Abstract Role</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>DeploymentAdmin</td>
<td>*.DeploymentAdmin</td>
<td>Allow</td>
</tr>
<tr>
<td>Deployment</td>
<td>DeploymentOwner</td>
<td>*.DeploymentAdmin</td>
<td>Allow</td>
</tr>
<tr>
<td>Deployment</td>
<td>Player</td>
<td>*.Player</td>
<td>Allow</td>
</tr>
<tr>
<td>Project</td>
<td>ProjectAdmin</td>
<td>*.ProjectAdmin</td>
<td>Allow</td>
</tr>
<tr>
<td>Project</td>
<td>ProjectOwner</td>
<td>*.ProjectOwner</td>
<td>Allow</td>
</tr>
</tbody>
</table>

**Column Description**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Shows whether the role mapping came from the <code>project-template.json</code> or <code>deployment-access-template.json</code> files.</td>
</tr>
<tr>
<td>Actual Role</td>
<td>Shows the name of the role resource definition with the mapping metadata.</td>
</tr>
<tr>
<td>Abstract Role</td>
<td>Shows the abstract roles (as specified on permission metadata) that map to the role.</td>
</tr>
<tr>
<td>Effect</td>
<td>Shows whether the permissions requested for the abstract role are allowed or denied.</td>
</tr>
</tbody>
</table>

### Improve Security with a Custom Domain Name

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. 
[Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

To improve security when connecting to the Cloud Canvas APIs, you can enforce a minimum Transport Layer Security (TLS) version and cipher suites. This can be achieved by creating a custom domain name through Amazon API Gateway and establishing TLS handshakes to Cloud Canvas APIs via service URLs that are generated with this custom domain.
Create a Custom Domain Name

For a video tutorial of these steps, see How do I define a custom domain name for my Amazon API Gateway API?

To create a custom domain name in API Gateway

1. Register an internet domain name.

You can register an internet domain using Amazon Route 53 or using a third-party domain registrar of your choice. An API's custom domain name can be the name of a subdomain or the root domain of a registered internet domain.

2. Request or import an SSL/TLS certificate.

Request an SSL/TLS certificate from AWS Certificate Manager (ACM), or import an SSL/TLS certificate into ACM. For more information, see Get Certificates Ready in AWS Certificate Manager.

3. Use the API Gateway console to create an edge-optimized custom domain name.

Select Edge-optimized for the endpoint type, and choose the appropriate TLS version. Then choose the ACM certificate you just requested. For more information, see How to Create an Edge-Optimized Custom Domain Name.

4. Create or update your DNS provider’s resource record.

After creating a custom domain name, you can check its details by choosing the domain name in the API Gateway console. Take note of the API Gateway domain name field, which is the default API endpoint. It has a format of name.cloudfront.net. Then create or update your DNS provider’s resource record to map your custom domain name to the default API endpoint. Without such a mapping, API requests bound for the custom domain name cannot reach API Gateway.

Create New Cloud Canvas Stacks Using the Custom Domain Name

Use the following steps to create new project and deployment stacks using your custom domain name.

To create new project and deployment stacks

1. To use the custom domain name for all service URLs in your Cloud Canvas project, specify the custom-domain-name argument when you create a new project stack using CLI commands.

The following command is an example.

```
lmbr_aws project create --stack-name SampleProject --region us-east-1 --custom-domain-name example.com
```
When the new project stack is created, you can create multiple deployments using normal CLI commands, without any additional change. All ServiceApi resources created within the project and deployment stacks generate a unique base path under the custom domain name, which is mapped to the actual API stage. Base paths have the format of `{region}.{stage}.{rest_api_id}`. You can find all these mappings in the API Gateway console.

<table>
<thead>
<tr>
<th>API</th>
<th>Stage</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>customdomain-ServiceApi</td>
<td>api</td>
<td>us-east-1-api/quxwH7y12</td>
</tr>
</tbody>
</table>

2. Verify local mappings files.

Service URLs generated from your project and deployment stacks have the format of `https://{custom_domain_name}/{region}.{stage}.{rest-api-id}`, which is also reflected in your local mappings files (`deploymentName.server.awsLogicalMappings.json`, `deploymentName.player.awsLogicalMappings.json`, and `user-settings.json`). To verify the correctness of the local mapping files, look for the URL for your service API. If the feature is enabled successfully, the service API URL should contain the custom domain name.

Clients can then use these custom service URLs to make their requests.

**Update Existing Cloud Canvas Stacks Using the Custom Domain Name**

Updating your old project and deployment stacks to use the custom domain name requires manual inputs and exposes the stacks to risk of rollback failure. New versions of custom resource handlers are created when you update the custom domain name, but they are not be used to process events because custom resources are locked by default to the version of the code that you used to create the resources. For more information, see Versioning of Custom Resources (p. 2303).

If necessary, you can override the locking of a custom resource to the version of the code with which it was created using the following steps.

**To update your existing project and deployment stacks**

1. Update your existing project stack using the `project update` command and the `custom-domain-name` argument.

   ```bash
lmbr_aws project update --custom-domain-name example.com
   ```

2. Set the `CustomResourceVersion` in your project template.

   a. Open the AWS Lambda console and find the custom resource handler for ServiceApi resources, which has a name like `SampleProject-CRH-CoreResourceTypes-Custom_ServiceApi`. Make a note of the new Lambda version number after the update.
b. Edit the definition of ServiceApi resource in `dev\Gems\CloudGemFramework\v1\ResourceManager\resource_manager\templates\project-template.json`. Add the metadata field and specify the custom resource version number.

```
"ServiceApi": {
  "Metadata": {
    "CloudCanvas": {
      "CustomResourceVersion": "2"
    }
  },
  "Properties": {
    ...
  },
  "Type": "Custom::ServiceApi",
  "DependsOn": [
    "CoreResourceTypes"
  ]
}
```

3. Update your project stack again using the same CLI command that you used in step 1, with the custom-domain-name argument specified. The specified version of the Lambda functions processes the resources this time and the custom domain name is used to generate service URLs.

Alternatively, you could have set CustomResourceVersion in the template to $LATEST before the first update to your project stack and only updated the project stack once. This practice is not recommended for environments where failed stack updates cannot be tolerated. Using $LATEST replicates the unsafe behavior in previous versions of Lumberyard where the most recent Lambda code version is used to process custom resource instance events.

4. Update your deployment stacks using the normal CLI commands without any additional change.

5. Update the resource templates defined in each enabled cloud gem. ServiceApi resources might be defined in each of these gems. Check the definition in `resource-template.json` under each gem and specify the same custom resource version number as you did for the project ServiceApi resource. You need to do something similar for any custom resource defined in your project or deployment stacks that uses service URLs directly, such as CrossGemCommunicationInterfaceResolver (defined in `dev\Gems\CloudGemFramework\v1\ResourceManager\resource_manager\config.py`). Find the custom resource handlers in the AWS Lambda console and update the resource definitions with CustomResourceVersion.

6. Update your deployment stacks again after the template updates. To do this, re-process the resources with the specified version of Lambda functions.

You can set CustomResourceVersion to $LATEST in the templates before updating the deployment stacks to avoid updating them twice as well, but this practice is not recommended for environments where failed stack updates cannot be tolerated.

7. Verify local mappings files.

Service URLs generated from your project and deployment stacks have the format of `https://{custom_domain_name}/{region}.{stage}.{rest-api-id}`, which is also be reflected in your local mappings files (`deploymentName.server.awsLogicalMappings.json`, `deploymentName.player.awsLogicalMappings.json`, and `user-settings.json`).

To verify the correctness of the local mapping files, look for the URL for your service API. If the feature is enabled successfully, the service API URL should contain the custom domain name.

Clients can then take advantage of these custom service URLs to make their requests.
Using the Cloud Canvas Command Line

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloud Canvas provides the `\dev\lmbr_aws\.cmd` command line tool for working with AWS resources. The tool invokes Python code that is located in the `\dev\Tools\lmbr_aws` directory.

Notes

- Lumberyard 1.9 renamed some `lmbr_aws` commands. For a list of the older commands and their newer equivalents, see Command Reorganization (p. 2412).
- For information on using Dynamic Content Cloud Gem `lmbr_aws` commands to update dynamic content, see Using the Dynamic Content CLI (p. 2169).
- For information on `lmbr_aws` commands for the cloud gem framework, see Using the Cloud Gem Framework Command Line (p. 2339).
- For information on `lmbr_aws` commands that manage roles and permissions, see Using the Cloud Canvas Command Line to Manage Roles and Permissions (p. 2381).

Syntax

```
lmbr_aws {command} {command-arguments}
```

{command} is one of commands in the command summary section that follows. {command-arguments} are the arguments accepted by the command. Arguments common to most commands are listed in the Common Arguments (p. 2391) section. Arguments unique to a command are listed in the detail section for the command.

Configuration

The tool gets its default AWS configuration from the same `~/.aws/credentials` and `~/.aws/config` files as the AWS command line tools (for information, see Configuring the AWS Command Line Interface). The `lmbr_aws` tool does not require that the AWS command line interface be installed.

Environment Variables

As with the AWS command line tools, the default AWS configuration can be overridden by using the following environment variables.

- AWS_ACCESS_KEY_ID The access key for your AWS account.
- AWS_SECRET_ACCESS_KEY The secret key for your AWS account.
- AWS_DEFAULT_REGION The default region to use; for example, `us-east-2`.
- AWS_PROFILE The default credential and configuration profile to use, if any.

Configuration Arguments

The following arguments can be used to override the AWS configuration from all other sources:

- `--assume-role role-name` or `-R role-name` – Specifies the IAM role to assume to perform the requested actions. The credentials in the `~/.aws/credentials` file must be able to assume the specified role.
• --aws-access-key \{access-key\} – The AWS access key that is used.
• --aws-secret-key \{secret-key\} – The AWS secret key that is used.
• --profile \{profile-name\} or -P \{profile-name\} – The AWS command line tool profile that is used.

Common Arguments

Most of the lmbr_aws commands accept the following arguments, in addition to their own individual arguments:

• --aws-directory \{aws\} – Identifies the \{game\}\AWS directory to use. The default is the value of the sys_game_folder property from \{root\}\bootstrap.cfg with AWS appended.
• --game-directory \{directory\} – Location of the game project directory. The default is \{root\}\{game\} where \{game\} is determined by the sys_game_folder setting in the \{root\}\bootstrap.cfg file.
• --help or -h – Display help for the command.
• --no-prompt – Suppresses calls that request input from the command prompt window.
• --region-override \{AWS_region\} – Specify a non-default AWS region to use in your local-project-settings.json file.
• --root-directory \{root\} – Identifies the lumberyard_installation\dev directory. The default is the current working directory.
• --user-directory \{user\} – Location of the user cache directory. The default is \{root\}\Cache \{game\}\AWS where \{game\} is determined by the sys_game_folder setting in the \{root\}\bootstrap.cfg file.
• --verbose – Shows additional output when executing commands.

Commands

Following are the lmbr_aws commands.

**cleanup Command**

The lmbr_aws --cleanup command deletes resources that you no longer need from your account in AWS.

**Warning**

Before you use the cleanup tool, be aware of the following points:

- Do not use the cleanup tool if you have a project stack name that begins with an IAM user name that you do not want to delete. Doing so can result in the deletion of the IAM user, its roles, and its profiles.
- When you delete an AWS resource, you permanently delete any objects that are stored in that resource. For example, if you delete an S3 bucket, all objects inside the bucket are also deleted.

--cleanup

Note that unlike other lmbr_aws commands, the cleanup command begins like a command argument, with two '--'. In the following example, replace \{cleanup-command-parameters\} with any of the valid cleanup arguments shown in the list that follows.

```
lmbr_aws --cleanup \{cleanup-command-parameters\}
```
In addition to the Common Arguments (p. 2391), the cleanup command accepts the following arguments:

- **--aws-access-key** `{aws-access-key}`
  The AWS access key to use.
- **--aws-secret-key** `{aws-secret-key}`
  The AWS secret key to use.
- **--delete-global-resources**
  Optional. If the --region argument is specified, deletes global resources such as IAM roles and Amazon S3 buckets. Ignored if --region is unspecified.
- **--confirm-deletion**
  Optional. Used with the --delete-global-resources command to confirm you understand that AWS resources will be deleted. Can be useful for automation.
- **--prefix** `{prefix [prefix...]}`
  Optional. Deletes stacks and Amazon S3 buckets in AWS that have the specified prefixes.
- **--except** `{exception [exception...]}`
  Optional. Do not delete resources that start with the specified exceptions. Can be useful, for example, when cleaning up old results while a test in progress.
- **--profile** `{profile}`
  Optional. The AWS profile to use. Defaults to the default AWS profile. AWS access and secret key take precedence over profile if set.
- **--region** `{region}`
  Optional. The AWS region to use. Defaults to us-east-1.

For more information on the cleanup tool, see Freeing Up AWS Resources (p. 2361).

**cloud-gem Commands**

The `lmbr_aws cloud-gem` commands create, enable, and disable cloud gems.

**cloud-gem create**

Create a cloud gem. This command is new in Lumberyard 1.11.

In addition to the Common Arguments (p. 2391), the `cloud-gem create` command accepts the following arguments:

- **--directory** `{path}`
  Optional. The directory where the gem is created. The default is `\dev\Gems\{name}\v{N}`, where `{name}` is the name that was specified by the --gem option and `{N}` is the major part of the gem version number as specified by the --version option.
- **--enable**
  Optional. Enable the gem for the current project when the gem is created. By default, the gem is not enabled when it is created.
- **--gem** `{name}`
  Required. The name of the gem to create.
Using the Command Line

• **--initial-content** *{initial-content-type}*

   Optional. Initialize the cloud gem's AWS directory with the contents specified by *{initial-content-type}*.

   Following are possible values for *{initial-content-type}***.

   **Note**
   Before you can use the api content options, do one of the following:
   • In Lumberyard Setup Assistant, choose **Compile the game code**
   • From a command prompt window on the \dev\Tools\LmbrSetup\Win directory, enter the following command:

   ```
   lmbr capabilities enable compilegame
   ```

• **api-lambda**

   Define a resource group with Amazon API Gateway and AWS Lambda function resources that conform to the Cloud Gem Framework's Service API pattern. You can add AWS resources as needed.

• **api-lambda-bucket**

   Define a resource group with API Gateway and Lambda function resources that conform to the Cloud Gem Framework's Service API pattern. Also creates an Amazon S3 bucket resource.

• **api-lambda-dynamodb**

   Define a resource group with API Gateway and Lambda function resources that conform to the Cloud Gem Framework's Service API pattern. Also creates an Amazon DynamoDB table resource.

• **bucket**

   Define a resource group that has an Amazon S3 bucket resource.

• **lambda**

   Define a resource group that has a Lambda function resource.

• **no-resources**

   Define a resource group that has no resources (except AccessControl (p. 2372), which is required). This is the default value.

• **resource-manager-plugin**

   Define a resource manager plugin.

• **--no-cpp-code**

   Optional. Define a gem that contains no C++ code and does not build as a .dll file. The gem can contain resource group definitions and/or resource manager plugins.

• **--version** *{version}*

   Optional. Set the gem version to the number specified by *{version}*. The default is 1.0.0. The version number must be in the format *{number}.{number}.{number}***.

**cloud-gem disable**

Disable a cloud gem that is in the current project. This command is new in Lumberyard 1.11.

In addition to the **Common Arguments (p. 2391)**, the cloud-gem disable command accepts the following arguments:

• **--gem** *{name}***
Required. The name of the gem to disable.

**cloud-gem enable**

Enable a cloud gem that is in the current project. This command is new in Lumberyard 1.11.

In addition to the Common Arguments (p. 2391), the `cloud-gem enable` command accepts the following arguments:

- `--gem {name}`

  Required. The name of the gem to enable.

### deployment Commands

The `lmbr_aws deployment` commands manage Cloud Canvas deployments.

**Note**

In Lumberyard 1.16 and later versions, the `deployment` commands install, modify, or remove gems in series rather than in parallel. This prevents issues that can occur in AWS when multiple resource groups are created, updated, or deleted at the same time. This default behavior can be overridden with the `--parallel` option, as in the following examples:

```shell
lmbr_aws deployment create --deployment deployment_name --parallel
lmbr_aws deployment update --parallel
lmbr_aws deployment delete --deployment deployment_name --parallel
```

**Warning**

Because it greatly increases the risk of errors when you modify your deployment, use of the `--parallel` option is not recommended.

**deployment create**

Create a complete and independent copy of all the resources needed by the Lumberyard project.

In addition to the Common Arguments (p. 2391), the `deployment create` command accepts the following arguments:

- `--deployment {deployment-name}` or `-d {deployment-name}`

  Required. The name of the deployment to create.

- `--confirm-aws-usage` or `-C`

  Optional. Confirms that you know that the `deployment create` command will create AWS resources for which you may be charged and that may perform actions that can affect permissions in your AWS account. Specify this argument to disable the related confirmation prompt.

**deployment default**

Set or show the default user and project deployments.

In addition to the Common Arguments (p. 2391), the `deployment default` command accepts the following arguments:

- `--set {deployment}`
Optional. Sets the default to the provided deployment name.

- **--clear**
  Optional. Clears the defaults.

- **--show**
  Optional. Shows the defaults.

- **--project**
  Optional. Applies --set and --clear to the project default instead of the user default. Ignored for --show.

Only one of the --set, --clear, and --show arguments is allowed.

If --set or --clear is specified, this command updates the `{root}\user\AWS\user-settings.json` file. If --project is provided, the `{root}\{game}\AWS\project-settings.json` file is updated.

deployment delete

Delete a complete and independent copy of all the resources needed by the Lumberyard project.

In addition to the Common Arguments (p. 2391), the deployment delete command accepts the following arguments:

- **--deployment {deployment-name}** or **-d {deployment-name}**
  Required. The name of the deployment to delete.

- **--confirm-resource-deletion**
  Optional. Acknowledges that the command will permanently delete the resources for the specified deployment. If this argument is not specified, a confirmation prompt is issued.

**Note**

AWS CloudFormation cannot delete stacks that define Amazon S3 buckets that contain data. To allow project stacks to be deleted, the project-template.json file specifies a DeletionPolicy of Retain for the configuration bucket. This causes AWS CloudFormation to not delete the bucket when the project stack is deleted. After the project stack has been deleted, the command removes all the objects from the configuration bucket and then deletes the bucket.

deployment list

List all deployments in the local project.

Example output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnotherDeployment</td>
<td>CREATE_PENDING</td>
<td>Resource is defined in the local project template but does not exist in AWS.</td>
</tr>
<tr>
<td>Development</td>
<td>CREATE_COMPLETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>03/04/16 18:43:11</td>
<td>arn:aws:cloudformation:us-</td>
</tr>
</tbody>
</table>
Using the Command Line

deployment list-resources

List all of the resources associated with the project.

In addition to the Common Arguments (p. 2391), the deployment list-resources command accepts the following arguments:

• --deployment {deployment-name} or -d {deployment-name}
  Optional. The name of the deployment to list resources for. If not specified, lists all the project's resources.

deployment protect

Marks a deployment as protected and issues a warning when a user (for example, a game developer, programmer, or tester) attempts to connected a development game client to live resources. For more information, see Using Protected Deployments (p. 2359).

In addition to the Common Arguments (p. 2391), the deployment protect command accepts the following arguments:

• --set {deployment-name}
  Optional. Specifies that the deployment is protected.

• --clear {deployment-name}
  Optional. Specifies that the deployment is not protected.

• --show
  Optional. Displays a list of the deployments that are currently protected.

  Note
  To display the protected status of deployments, you can also use either the deployment list (p. 2395) or mappings list (p. 2401) command.

deployment release

Sets, shows, or removes the release deployment.

In addition to the Common Arguments (p. 2391), the deployment release command accepts the following arguments:

• --set {deployment-name}
  Required. The name of the deployment to set as the release.

• --clear
  Removes the release designation from the current release deployment.

• --show
Shows the deployment that is currently configured as the release deployment.

**deployment tags**

Manages the tags for a deployment. Deployment tags specify resource group overrides for a deployment.

In addition to the Common Arguments (p. 2391), the `deployment tags` command accepts the following arguments:

- `--add {tag} [{tag} ...]`
  
  Optional. Specifies the tags to add to a deployment. The tags are saved in the `lumberyard_version\dev\project_name\AWS\local-project-settings.json` file.
- `--clear`
  
  Optional. Clears all tags for a deployment.
- `--delete {tag} [{tag} ...]`
  
  Optional. Specifies the tags to delete from a deployment.
- `--deployment {deployment-name} or -d {deployment-name}`
  
  Optional. The name of the deployment.
- `--list`
  
  Optional. Lists all the tags for a deployment.

**deployment update-access**

Sets, shows, or removes the release deployment.

In addition to the Common Arguments (p. 2391), the `deployment update-access` command accepts the following arguments:

- `--confirm-aws-usage or -C`
  
  Optional. Confirms that you know this command creates AWS resources for which you can be charged and that it may perform actions that can affect permissions in your AWS account. Specify this argument to disable the related confirmation prompt.
- `--confirm-resource-deletion`
  
  Optional. If the operation will delete resources permanently, confirms your acknowledgement and approval. If this argument is not specified, you are prompted to confirm completion of the operation. Specify this argument to disable the related confirmation prompt.
- `--confirm-security-change`
  
  Optional. Confirms that you know that this command may make security-related changes. Specify this argument to disable the related confirmation prompt.
- `--deployment {deployment-name} or -d {deployment-name}`
  
  Optional. The name of the deployment whose access stack is updated. If omitted, the default deployment is updated. Use * to update all deployments.

**deployment update, deployment upload, deployment upload-resources**

Updates a deployment. `deployment upload` or `deployment upload-resources` can be used instead of `deployment update`.
In addition to the Common Arguments (p. 2391), the deployment update command accepts the following arguments:

- **--confirm-aws-usage or -C**
  
  Optional. Confirms that you know this command creates AWS resources for which you can be charged and that it may perform actions that can affect permissions in your AWS account. Specify this argument to disable the related confirmation prompt.

- **--confirm-resource-deletion**
  
  Optional. If the operation will delete resources permanently, confirms your acknowledgement and approval. If this argument is not specified, you are prompted to confirm completion of the operation. Specify this argument to disable the related confirmation prompt.

- **--confirm-security-change**
  
  Optional. Confirms that you know that this command may make security-related changes. Specify this argument to disable the related confirmation prompt.

- **--deployment {deployment-name} or -d {deployment-name}**
  
  Optional. The name of the deployment to update. If omitted, the default deployment is updated.

**function Commands**

The lmbr_aws function commands manage folders, CloudWatch logs, and code for Lambda functions.

**function create-folder**

Recreates the default function folder for a Lambda function resource.

In addition to the Common Arguments (p. 2391), the function create-folder command accepts the following arguments:

- **--function {function-name} or -f {function-name}**
  
  Required. The logical name of a Lambda function resource.

- **--resource-group {resource-group-name} or -r {resource-group-name}**
  
  Required. The name of a resource group.

- **--force**
  
  Optional. Skips checks for existence of resource and type of resource. Used when creating folders for functions not yet created.

**function get-log**

Retrieves data from a CloudWatch Logs log file.

In addition to the Common Arguments (p. 2391), the function get-log command accepts the following arguments:

- **--deployment {deployment-name} or -d {deployment-name}**
  
  Optional. The name of a deployment. If this argument is specified, the --resource-group argument must also be specified. If this argument is omitted, then the function must exist in the project stack.

- **--function {function-name} or -f {function-name}**
Required. The logical name of a Lambda function resource.

- **--resource-group** `{resource-group-name}` or `-r` `{resource-group-name}`
  
  Optional. The name of a resource group. If specified, the `--deployment` argument must also be specified.

- **--log-stream-name** `{log-stream-name}` or `-l` `{log-stream-name}`
  
  Optional. The log stream name or partial log stream name. If omitted, the most recent log stream is shown.

**function upload-code**

Updates Lambda functions for a deployment or project.

In addition to the Common Arguments (p. 2391), the `function upload-code` command accepts the following arguments:

- **--deployment** `{deployment-name}` or `-d` `{deployment-name}`
  
  Optional. The name of a deployment to update. If neither this argument nor the `--project` argument is specified, the default deployment is updated. If the `--project` argument is specified, the `--deployment` argument is ignored.

- **--function** `{function-name}`
  
  Optional. The name of the Lambda function to update. If not specified, all Lambda functions in the resource group are updated.

- **--keep**
  
  Optional. Keep the generated `.zip` file instead of deleting it after it is uploaded.

- **--project**
  
  Optional. Updates the project. Overrides the `--deployment` argument.

- **--resource-group** `{resource-group-name}` or `-r` `{resource-group-name}`
  
  Optional. The name of a resource group that has the function to update.

**login-provider Commands**

The `lmbr_aws login-provider` commands add, remove, and update social network login providers in Amazon Cognito identity pool configuration.

**login-provider add**

Add a player login provider to the Amazon Cognito identity pool configuration. Login providers allow your game's players to log in using their social network identity, such as Facebook or using their Amazon identity. For more information, see Controlling Access to Resources (p. 2370).

In addition to the Common Arguments (p. 2391), the `login-provider add` command accepts the following arguments:

- **--provider** `{provider-name}`
  
  Required. The name of the provider. The name must be `amazon`, `google` or `facebook`, or, if you are using a generic OpenID provider, a name that you choose.

- **--app-id** `{application-id}`
Using the Command Line

Required. The application id from your login provider (this is usually different from your client ID).

• --client-id {client-id}

Required. The unique application client ID for the login provider.

• --client-secret {client-secret}

Required. The secret key to use with your login provider.

• --redirect-uri {redirect-uri}

Required. The redirect URI to use with your login provider.

• --provider-uri {provider-uri}

Optional. The URI for a generic open ID connect provider. This is only used for generic OpenID providers.

• --provider-port {provider-port}

Optional. The port your provider listens on for its API. This is only used for generic OpenID providers.

• --provider-path {provider-path}

Optional. The path portion of your provider's URI. This is only used for generic OpenID providers.

This command saves its configuration in a `player-access/auth-settings.json` object in the project's configuration bucket so that the `PlayerAccessTokenExchange` Lambda function can access it.

**Note**
You must run `project upload` after running this command so that the `PlayerAccessIdentityPool` configuration is updated to reflect the change.

**login-provider remove**

Remove a player login provider from the Amazon Cognito identity pool configuration.

In addition to the Common Arguments (p. 2391), the `login-provider remove` command accepts the following argument:

• --provider {provider-name}

Required. The name of the provider.

The `login-provider remove` command saves the configuration in a `/player-access/auth-settings.json` object in the project's configuration bucket so that the `PlayerAccessTokenExchange` Lambda function can access it.

**Note**
You must run `lmbr_aws project upload` after running this command so that the `PlayerAccessIdentityPool` configuration is updated to reflect the change.

**login-provider update**

Update a player login provider in the Amazon Cognito identity pool configuration. Login providers allow your game's players to log in using their social network identity, such as Facebook, or using their Amazon identity. For more information, see Controlling Access to Resources (p. 2370).

In addition to the Common Arguments (p. 2391), the `login-provider update` command accepts the following arguments:

• --provider {provider-name}
Required. The name of the updated provider. The name must be `amazon`, `google` or `facebook`, or, if you are using a generic OpenID provider, the name that you chose when the provider was added.

- `--app-id {application-id}`

Optional. The application ID from your login provider (this is usually different from your client ID).

- `--client-id {client-id}`

Optional. The unique application client ID for the login provider.

- `--client-secret {client-secret}`

Optional. The secret key to use with your login provider.

- `--redirect-uri {redirect-uri}`

Optional. The redirect URI to use with your login provider.

- `--provider-uri {provider-uri}`

Optional. The URI for a generic open id connect provider. This argument is used only for generic OpenID providers.

- `--provider-port {provider-port}`

Optional. The port the provider listens on for the provider's API. This argument is used only for generic OpenID providers.

- `--provider-path {provider-path}`

Optional. The path portion of the provider's URI. This argument is used only for generic OpenID providers.

The `login-provider update` command saves its configuration in a `/player-access/auth-settings.json` object in the project’s configuration bucket so that the `PlayerAccessTokenExchange` Lambda function can access it.

**Note**

You must run `lmbr_aws project upload` after running this command so that the `PlayerAccessIdentityPool (p. 2239)` configuration is updated to reflect the change.

**mappings Commands**

The `lmbr_aws mappings` commands manage the mappings of logical to physical resource names in a Cloud Canvas deployment.

**mappings list**

Show the logical to physical resource name mappings.

**Example output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelloWorld.SayHello</td>
<td>AWS::Lambda::Function</td>
<td>foo-hw-Development-</td>
</tr>
<tr>
<td>ZDLXUB7FK94-HelloWo-SayHello-1FADMNE5MCO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PlayerAccessIdentityPool</td>
<td>Custom::CognitoIdentityPool</td>
<td>us-east-2:1:208f6d6a-</td>
</tr>
<tr>
<td>f929-4121-9947-a03169b9582e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PlayerLoginIdentityPool</td>
<td>Custom::CognitoIdentityPool</td>
<td>us-east-2:2:3020e175-6d2-4860-8dad-1d657162cbb2</td>
</tr>
<tr>
<td>PlayerAccessTokenExchange</td>
<td>AWS::Lambda::Function</td>
<td>foo-hw-</td>
</tr>
<tr>
<td>account_id</td>
<td>Configuration</td>
<td>&lt;ACCOUNTID&gt;</td>
</tr>
</tbody>
</table>
**mappings update**

Update the friendly name to physical resource ID mappings for the current default deployment or the release deployment.

In addition to the Common Arguments (p. 2391), the `mappings update` command accepts the following arguments:

- **--release**
  
  Optional. Causes the release mappings to be updated. By default, only the mappings used when launching the game from inside the editor are updated.

  The command looks in the `resource-template.json` file for `Metadata.CloudCanvas.PlayerAccess` properties on resource definitions. It then queries AWS CloudFormation for the physical names of those resources in the current default deployment. If the `--release` argument is specified, the release deployment is queried.

- **--deployment** `{deployment-name}` or `-d` `{deployment-name}`

  Optional. Exports client and server mapping files for the specified deployment to the `\dev\project_name\Config` directory in the format `{deployment_name}.player.awsLogicalMappings.json` and `{deployment_name}.server.awsLogicalMappings.json`. The `{deployment-name}` argument is required and is case-sensitive.

  When you run a game launcher such as the one at `\Bin64\NNN\CloudGemSamplesLauncher.exe`, you can use the `-cc_override_resource_map `{mappings-name}` argument to choose the mapping. For more information, see Selecting a Deployment with a PC Launcher (p. 2358).

**parameter Commands**

The `lmbr_aws parameter` commands manage parameters for Cloud Canvas deployments and resource groups.

**parameter clear**

Clears the specified parameter configuration for your project. The project must be initialized (a project stack must have been created) before you can clear parameters.

In addition to the Common Arguments (p. 2391), the `parameter clear` command accepts the following arguments:

- **--deployment** `{deployment-name}` or `-d` `{deployment-name}`

  Optional. Clears the parameter value for the specified deployment. `{deployment-name}` can be *, in which case the parameter value used for all deployments that do not override the value is cleared. If omitted, the parameter value is cleared for all deployments, including *.

- **--resource-group** `{resource-group-name}` or `-r` `{resource-group-name}`

  Optional. Clears the parameter value for the specified resource-group. `{resource-group-name}` can be *, in which case the parameter value used for all resource groups that do not override the value is cleared. If omitted, the parameter value is cleared for all resource groups, including *.

- **--parameter** `{parameter-name}` or `-p` `{parameter-name}`

  Required. The parameter to clear.
parameter list

Lists the parameters currently configured for your project. The project must be initialized (a project stack must have been created) before you can list parameters.

In addition to the Common Arguments (p. 2391), the parameter list command accepts the following arguments:

- `--deployment {deployment-name}` or `-d {deployment-name}`
  Required. Limits the list to the specified deployment. `{deployment-name}` can be *, in which case parameters that apply to all deployments are listed.
- `--resource-group {resource-group-name}` or `-r {resource-group-name}`
  Required. Limits the list to the specified resource group. `{resource-group-name}` can be *, in which case parameters that apply to all resource groups are listed.
- `--parameter {parameter-name}` or `-p {parameter-name}`
  Optional. Limits the list to the specified parameter.

parameter set

Sets parameter configuration for your project. The project must be initialized (a project stack must have been created) before you can set parameters.

In addition to the Common Arguments (p. 2391), the parameter set command accepts the following arguments:

- `--deployment {deployment-name}` or `-d {deployment-name}`
  Required. Sets the parameter value for the specified deployment. `{deployment-name}` can be *, in which case the parameter value is used for all deployments that do not override the value.
- `--resource-group {resource-group-name}` or `-r {resource-group-name}`
  Required. Sets the parameter value for the specified resource group. `{resource-group-name}` can be *, in which case the parameter value is used for all resource groups that do not override the value.
- `--parameter {parameter-name}` or `-p {parameter-name}`
  Required. Specifies the parameter whose value will be set.
- `--value {parameter-value}` or `-v {parameter-value}`
  Required. Specifies the value to set.

profile Commands

The lmb_aws profile commands manage the AWS profiles that you use for Cloud Canvas.

profile add

Add an AWS profile to the AWS command line tool configuration.

In addition to the Common Arguments (p. 2391), the profile add command accepts the following arguments:

- `--aws-access-key {accesskey}`
  Required. The AWS access key associated with the added profile.
• --aws-secret-key \(\text{secretkey}\)
  Required. The AWS secret key associated with the added profile.
• --profile \(\text{profile-name}\) or -P \(\text{profile-name}\)
  Required. The name of the AWS profile to add.
• --make-default
  Optional. Make the new profile the default profile.

profile default

Set, clear, or show the default AWS profile. The profile default command modifies the DefaultProfile section of the user-settings.json (p. 2237) file. As of Lumberyard 1.15, the Cloud Canvas cleanup tool (p. 2362) uses this setting to determine the AWS profile to use.

In addition to the Common Arguments (p. 2391), the profile default command accepts the following arguments:
• --set \(\text{profile-name}\)
  Optional. Set the default profile for the current deployment to the provided AWS profile name.
• --clear
  Optional. Clear the default profile.
• --show
  Optional. Show the default profile.

profile list

List the AWS profiles that have been configured.

profile remove

Remove an AWS profile from the AWS command line configuration.

In addition to the Common Arguments (p. 2391), the profile remove command accepts the following argument:
• --profile \(\text{profile-name}\) or -P \(\text{profile-name}\)
  Required. The name of the AWS profile to remove.

profile rename

Rename an AWS profile in the AWS command line tool configuration.

In addition to the Common Arguments (p. 2391), the profile rename command accepts the following arguments:
• --old \(\text{old-profile-name}\)
  Required. The name of the AWS profile to change.
• --new \(\text{new-profile-name}\)
  Required. The new name of the AWS profile.
profile update

Update an AWS profile.

In addition to the Common Arguments (p. 2391), the profile update command accepts the following arguments:

• `--aws-access-key {accesskey}`
  Optional. The AWS access key associated with the updated profile. The default is to not change the AWS access key associated with the profile.

• `--aws-secret-key {secretkey}`
  Optional. The AWS secret key associated with the updated profile. The default is to not change the AWS secret key associated with the profile.

• `--profile {profilename}` or `-P {profile-name}`
  Required. The name of the AWS profile to update.

Note

To make an existing profile the default profile, use the profile default --set {profile-name} (p. 2404) command.

project Commands

The lmbr_aws project commands manage Cloud Canvas projects in Lumberyard.

project create

Initialize Cloud Canvas resource management for a Lumberyard project. This includes creating a set of default Resource Definitions (p. 2236) in the dev\game\AWS directory and a AWS CloudFormation stack that contains the resources that the Cloud Canvas resource manager uses to manage your game resources.

In addition to the Common Arguments (p. 2391), the project create command accepts the following arguments:

• `--stack-name {stack-name}`
  Optional. The name used for the project's AWS CloudFormation stack. The default is the name of the {game} directory.

• `--confirm-aws-usage` or `-C`  
  Optional. Confirms that you know this command will create AWS resources for which you may be charged and that it may perform actions that can affect permissions in your AWS account. Specify this argument to disable the related confirmation prompt.

• `--files-only`
  Optional. Writes the default configuration data to the {game}\AWS directory and exits. The directory must be empty or must not exist.

• `--region {region}`
  Required. The AWS region in which the project stack will be created.

Note

The region argument can be used only with the project create and resource-importer list-importable-resources commands. To manually override the AWS region in other lmbr_aws commands, use the --region-override argument.
• `--admin-roles`
  
  Optional. Creates the ProjectOwner and ProjectAdmin roles. The default is to create these roles unless `--no-admin-roles` argument is used. For more information about these roles, see Cloud Canvas Built-In Roles and Policies (p. 2373).

• `--no-admin-roles`
  
  Optional. Turns off the creation of the ProjectOwner and ProjectAdmin roles. This is recommended unless you have a need for these roles.

How `project create` Works

The `project create` command performs the following tasks:

1. Creates the project's AWS CloudFormation stack using a bootstrap template that defines only the Configuration Bucket (p. 2248) resource.
2. Uploads the `project-template.json` (p. 2238) file and the zipped up contents of the `project-code` subdirectory (p. 2240) to the Configuration Bucket (p. 2248).
3. Uses the uploaded `project-template.json` file to perform an AWS CloudFormation stack update operation. The `project-code.zip` file is used to create the Lambda function resources defined by the `project-template.json` file.

Note

• If the `{root}\{game}\AWS` directory is empty or does not exist, `project create` creates the directory if necessary and copies the contents of the `dev\Tools\lmbr_aws\AWSResourceManager\default-project-content` directory to that directory.
• `project create` fails if a stack with the specified name already exists in the configured AWS account and region. In this case you can use the `--stack` argument to specify a different name for the project stack.
• `project create` fails if the `dev\{game\}AWS\local-project-settings.json` file has a non-empty ProjectStackId property. Initially, the ProjectStackId property is not present in the `local-project-settings.json` file. After the project stack is created in step 1, the ProjectStackId property is written to the file and is the project's AWS CloudFormation stack ID.
• If the stack update process in step 2 fails on the first attempt, you can retry by using the `project upload` command.

`project create-extension-template`

Creates an extension template for adding project, deployment, or deployment access resources that complement the existing `project-template.json`, `deployment-template.json`, or `deployment-access-template.json` files.

In addition to the Common Arguments (p. 2391), the `project create-extension-template` command accepts the following arguments:

• `--project`
  
  Optional. Creates a `project-template-extensions.json` file in the project directory.
• `--deployment`
  
  Optional. Creates a `deployment-template-extensions.json` file in the project directory.
• `--deployment-access`
Optional. Creates a deployment-access-template-extensions.json file in the project directory.

**project delete**

Delete the AWS CloudFormation stack that contains your project's resources. You must delete all of the project's deployments before deleting the project stack. After deleting the project stack, you must create a new project stack before you can use AWS CloudFormation resource manager for your project.

In addition to the Common Arguments (p. 2391), the `project delete` command accepts the following argument:

- `--confirm-resource-deletion`

Optional. Confirms your acknowledgement and approval that the operation will delete resources permanently. If this argument is not specified, you will be prompted to confirm completion of the operation. Specify this argument to disable the related confirmation prompt.

AWS CloudFormation cannot delete stacks that define Amazon S3 buckets that contain data. To allow project stacks to be deleted, the `project-template.json` file specifies a `DeletionPolicy` of `Retain` for the configuration bucket. This causes AWS CloudFormation to not delete the bucket when the project stack is deleted. After the project stack has been deleted, the command removes all the objects from the configuration bucket and then deletes the bucket.

**Note**

Before you use the `project delete` command, ensure that the default AWS credentials specified by the `~/.aws/credentials` file on the local file system are the same as the credentials that are specified in the Cloud Canvas Credentials Manager. If the credentials are not the same, the project might not delete properly. You can use the Cloud Canvas Cleanup Tool (p. 2363) to remove orphaned resources after a failed project stack deletion.

**project list-resources**

List all of the resources associated with the project.

**project update, project upload**

Update the project's AWS CloudFormation stack. `project upload` can be used instead of `project update`.

In addition to the Common Arguments (p. 2391), the `project update` command accepts the following arguments:

- `--confirm-aws-usage` or `-C`

Optional. Confirms that you know this command will create AWS resources for which you may be charged and that it may perform actions that can affect permission in your AWS account. Specify this argument to disable the related confirmation prompt.

- `--confirm-resource-deletion`

Optional. If the operation will delete resources permanently, confirms your acknowledgement and approval. If this argument is not specified, you are prompted to confirm completion of the operation. Specify this argument to disable the related confirmation prompt.

**How project update works**

The project update command performs the following tasks:
1. Uploads the `project-template.json` file and the zipped up contents of the `project-code subdirectory` to the Configuration Bucket.

2. Uses the uploaded `project-template.json` file to perform an AWS CloudFormation stack update operation. The `project-code.zip` file is used to create the Lambda function resources defined by the `project-template.json` file.

**Note**

The `project update` command fails if the `dev\{game}\AWS\local-project-settings.json` file does not exist or does not have a valid `ProjectStackId` property.

### project update-framework-version

Updates the Cloud Gem framework version for a project.

In addition to the Common Arguments (p. 2391), the `project update-code` command accepts the following arguments:

- `--confirm-aws-usage` or `-C` (Optional. Confirms that you know that the deployment create command will create AWS resources for which you may be charged and that may perform actions that can affect permissions in your AWS account. Specify this argument to disable the related confirmation prompt.)
- `--confirm-resource-deletion` (Optional. Acknowledges that the command will permanently delete the resources belonging to the specified deployment. If this argument is not specified, a confirmation prompt is issued.)
- `--confirm-security-change` (Optional. Confirms that you know that this command may make security-related changes. Specify this argument to disable the related confirmation prompt.)

For more information, see Applying Framework Updates to a Project (p. 2343).

### resource-group Commands

The `lmbr_aws resource-group` commands manage the resource groups in your Cloud Canvas project.

#### resource-group add

**Note**

The functionality of this command has changed in Lumberyard 1.11. This command is deprecated in favor of the `resource-group enable (p. 2409)` and `cloud-gem create (p. 2392)` commands. As of Lumberyard 1.11, project local resource groups are deprecated in favor of cloud gems. Because cloud gems are reusable, we recommend that you use them instead of local resource groups.

As of Lumberyard 1.11, the `resource-group add` command has the following behavior:

- If the resource group specified in the `--resource-group` parameter exists but is disabled, the resource group is enabled. This is the equivalent of the `lmbr_aws resource-group enable` command.
- If the resource group does not exist, the `lmbr_aws cloud-gem create` command is called to create a new resource group in a cloud gem. The name of the cloud gem is specified by the `--resource-group {resource-group-name}` parameter.
- By default, the new gem has no resources. This corresponds to the `cloud-gem create` command's `--initial-content no-resources` option.
- If the `--include-example-resources` option is specified, the equivalent of the `cloud-gem create` command's `--initial-content api-lambda-dynamodb` option is called.
In addition to the Common Arguments (p. 2391), the `resource-group add` command accepts the following arguments:

- `--resource-group {resource-group-name} or -r {resource-group-name}`
  Required. The name of an existing resource group to enable. If no resource group with the specified name exists, specifies the name of the cloud gem and resource group to be created.
- `--gem {gem-path}`
  Optional. Looks for the resource group definition at `Gems\group\AWS` or `gem-path\AWS`.
- `--include-example-resources`
  Optional. The equivalent of the `--initial-content api-lambda-dynamodb` option of the `cloud-gem create (p. 2392)` command.

**resource-group disable**

Disable a resource group for debugging. This command is new in Lumberyard 1.11.

In addition to the Common Arguments (p. 2391), the `resource-group disable` command accepts the following argument:

- `--resource-group {resource-group-name} or -r {resource-group-name}`
  Required. The name of the resource group to disable.

By default, when a cloud gem is enabled for a project, all its resource groups are enabled. Disabling a resource group has the following consequences:

- When a deployment is created, the resources that the disabled resource group defines are not created in AWS.
- When a deployment is updated, the resources that were defined by the disabled resource group are removed.
- The resource group is added to a list of disabled resource groups in the project's `local-project-settings.json` file.

**Note**

Prior to Lumberyard 1.11, the `local-project-settings.json` file kept a list of enabled resource groups. By default, all of a cloud gem's resource groups are enabled when the cloud gem is enabled. Therefore, listing only the disabled resource groups makes it easier to identify them.

Usually it is better to disable the cloud gem that provides a resource group than to disable the resource group itself. Therefore, we recommend that you use the `resource-group disable` command only for debugging or testing. To reenable a resource group, you can use the `lmbr_aws resource-group enable` command.

**resource-group enable**

Enable a resource group. This command is new in Lumberyard 1.11.

In addition to the Common Arguments (p. 2391), the `resource-group disable` command accepts the following argument:

- `--resource-group {resource-group-name} or -r {resource-group-name}`
  Required. The name of the resource group to enable.
resource-group list

List all the resource groups found in the local deployment template and in the selected deployment in AWS.

In addition to the Common Arguments (p. 2391), the resource-group list command accepts the following argument:

• **--deployment** `{deployment-name}` or **-d** `{deployment-name}`
  
  Optional. The name of the deployment to list resource groups for. If not given, the default deployment is used.

Example output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnotherResourceGroup</td>
<td>CREATE_PENDING</td>
<td>Resource is defined in the local deployment template but does not exist in AWS.</td>
</tr>
<tr>
<td>HelloWorld</td>
<td>CREATE_COMPLETE</td>
<td></td>
</tr>
</tbody>
</table>

resource-group list-resources

List all of the resources associated with the project.

In addition to the Common Arguments (p. 2391), the resource-group list-resources command accepts the following arguments:

• **--deployment** `{deployment-name}` or **-d** `{deployment-name}`
  
  Optional. The name of the deployment to list resources for. If not specified, lists all the project's resources.

• **--resource-group** `{resource-group-name}` or **-r** `{resource-group-name}`
  
  Optional. The name of the resource group to list resources for. If specified, deployment must also be specified. If not specified, all deployment or project resources are listed.

resource-group remove

**Note**

The functionality of this command has changed in Lumberyard 1.11. This command is deprecated in favor of the cloud-gem disable (p. 2393) and resource-group disable (p. 2409) commands.

As of Lumberyard 1.11, the resource-group remove command has the following behavior:

• If the resource group specified in the **--resource-group** parameter is provided by a cloud gem, disables the cloud gem. This is the equivalent of the lmbr_aws cloud-gem disable command.

• If the resource group is not provided by a cloud gem, disables the resource group. This is the equivalent of the lmbr_aws resource-group disable command.
In addition to the Common Arguments (p. 2391), the resource-group remove command accepts the following argument:

- `--resource-group {resource-group-name} or -r {resource-group-name}`

  Required. The name of the resource group or cloud gem to be disabled.

**resource-group upload-resources (upload, update)**

Update a resource group's AWS CloudFormation stack in a deployment. resource-group upload or resource-group update can be used instead of resource-group upload-resources.

In addition to the Common Arguments (p. 2391), the resource-group upload command accepts the following arguments:

- `--confirm-aws-usage or -C`

  Optional. Confirms that you know this command will create AWS resources for which you may be charged and that it may perform actions that can affect permissions in your AWS account. Specify this argument to disable the related confirmation prompt.

- `--confirm-resource-deletion`

  Optional. If the operation will delete resources permanently, confirms your acknowledgement and approval. Specify this argument to disable the related confirmation prompt.

- `--confirm-security-change`

  Optional. Confirms that you know that this command may make security-related changes. Specify this argument to disable the related confirmation prompt.

- `--deployment {deployment-name} or -d {deployment-name}`

  Optional. The name of the deployment to update. If not specified, the default deployment is updated.

- `--resource-group {resource-group-name}, or -r {resource-group-name}`

  Required. The name of the resource group to update. If not specified, all resource groups in the deployment are updated.

**resource-importer Commands**

The lmbr_aws resource-importer commands list and import AWS resource definitions to your Cloud Canvas project.

**resource-importer import-resource**

Import a resource to a resource group.

In addition to the Common Arguments (p. 2391), the resource-importer import-resource command accepts the following arguments:

- `--type {dynamodb|s3|lambda|sns|sqs}`

  Optional. The type of the AWS resource to import. Choose from dynamodb, s3, lambda, sns or sqs.

- `--arn ARN`

  Required. The ARN of the AWS resource to import.

- `--resource-name {resource-name}`

  Required. The name of the resource to import.
• **--resource-group** `{resource-group-name}` or `-r {resource-group-name}`
  Required. The name of the resource group to import.
• **--download**
  Optional. If specified, downloads the contents of the Amazon S3 bucket.

`resource-importer list-importable-resources`

List all supported resources currently existing on AWS.

In addition to the Common Arguments (p. 2391), the `resource-importer list-importable-resources` command accepts the following arguments:

• **--type** `{dynamodb|s3|lambda|sns|sqs}`
  Required. The type of the AWS resource to list. Choose from dynamodb, s3, lambda, sns or sqs.
• **--region** `{region}`
  Optional. The AWS region of the resources. The default value is the region of the project stack, if it exists.

  **Note**
  The region argument can be used only with the `resource-importer list-importable-resources` and `project create` commands. To manually override the AWS region in other `lmbr_aws` commands, use the --region-override argument.

### Command Reorganization

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

As of Lumberyard 1.9, the commands made available by the `lmbr_aws` command line tool have been reorganized. The previous commands are still functional but are subject to removal in a future release.

The following table shows how the previous commands map to the new commands. In a few cases single commands have been split into two or more commands for clarity and ease of use. These are commented accordingly. For information about the commands, see Using the Cloud Canvas Command Line (p. 2390).

<table>
<thead>
<tr>
<th>Previous Command</th>
<th>New Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>add#login#provider</td>
<td>login#provider add</td>
</tr>
<tr>
<td>add#profile</td>
<td>profile add</td>
</tr>
<tr>
<td>add#resource#group</td>
<td>resource#group add</td>
</tr>
<tr>
<td>clear#parameter</td>
<td>parameter clear</td>
</tr>
<tr>
<td>create#deployment</td>
<td>deployment create</td>
</tr>
<tr>
<td>create#project#project</td>
<td>project create</td>
</tr>
<tr>
<td>default#deployment</td>
<td>deployment default</td>
</tr>
<tr>
<td>Previous Command</td>
<td>New Command</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>default#profile</td>
<td>profile default</td>
</tr>
<tr>
<td>delete#deployment</td>
<td>deployment delete</td>
</tr>
<tr>
<td>delete#project</td>
<td>project delete</td>
</tr>
<tr>
<td>get#function#log</td>
<td>function get#log</td>
</tr>
<tr>
<td>import#resource</td>
<td>resource#importer import#resource</td>
</tr>
<tr>
<td>list#deployment</td>
<td>deployment list</td>
</tr>
<tr>
<td>list#importable#resource</td>
<td>list#importable#resources</td>
</tr>
<tr>
<td>list#mappings</td>
<td>mappings list</td>
</tr>
<tr>
<td>list#parameters</td>
<td>parameter list</td>
</tr>
<tr>
<td>list#profiles</td>
<td>profile list</td>
</tr>
<tr>
<td>list#resource#group</td>
<td>resource#group list</td>
</tr>
<tr>
<td>list#resources</td>
<td>project list#resources</td>
</tr>
<tr>
<td></td>
<td>deployment list#resources</td>
</tr>
<tr>
<td></td>
<td>resource#group list#resources</td>
</tr>
<tr>
<td>The list#resources command supported # deployment and # resource#group arguments.</td>
<td></td>
</tr>
<tr>
<td>The project list#resources command is the same as list#resources without the # deployment and # resource#group arguments.</td>
<td></td>
</tr>
<tr>
<td>The deployment list#resources command is the same as list#resources with only the # deployment argument.</td>
<td></td>
</tr>
<tr>
<td>The resource#group list#resources command is the same as list#resources with both the # deployment and # resource#group arguments.</td>
<td></td>
</tr>
<tr>
<td>protect#deployment</td>
<td>deployment protect</td>
</tr>
<tr>
<td>release#deployment</td>
<td>deployment release</td>
</tr>
<tr>
<td>remove#login#provider</td>
<td>provider remove</td>
</tr>
<tr>
<td>remove#profile</td>
<td>profile remove</td>
</tr>
<tr>
<td>remove#resource#group</td>
<td>resource#group remove</td>
</tr>
<tr>
<td>rename#profile</td>
<td>profile rename</td>
</tr>
<tr>
<td>set#parameter</td>
<td>parameter set</td>
</tr>
<tr>
<td>update#deployment#access</td>
<td>deployment update # access</td>
</tr>
<tr>
<td>update#login#provider</td>
<td>provider update</td>
</tr>
<tr>
<td>Previous Command</td>
<td>New Command</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>update#mappings</td>
<td>mappings update</td>
</tr>
<tr>
<td>update#profile</td>
<td>profile update</td>
</tr>
<tr>
<td>update#project#code</td>
<td>project code update</td>
</tr>
<tr>
<td>update#project</td>
<td>project upload</td>
</tr>
</tbody>
</table>

You can use `project update` instead of `project upload`.

<table>
<thead>
<tr>
<th>upload#lambda#code</th>
<th>function upload#code</th>
</tr>
</thead>
<tbody>
<tr>
<td>upload#resource</td>
<td>deployment upload</td>
</tr>
<tr>
<td>resource#group</td>
<td>upload#resources</td>
</tr>
</tbody>
</table>

The `upload#resources` command supported a `##resource#group` argument.

The `deployment upload` command is the same as `upload#resources` without the `##resource#group` argument.

The `resource#group upload` command is the same as `upload#resources` with the `##resource#group` argument.

You can use `deployment update` and `resource#group update` instead of `deployment upload` and `resource#group upload`.

**Argument Aliases**

As a convenience, aliases have been added for the following common arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Alias Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>##assume#role</code></td>
<td><code>#R</code></td>
</tr>
<tr>
<td><code>##confirm#aws#usage</code></td>
<td><code>#C</code></td>
</tr>
<tr>
<td><code>##deployment</code></td>
<td><code>#d</code></td>
</tr>
<tr>
<td><code>##function</code></td>
<td><code>#f</code></td>
</tr>
<tr>
<td><code>##log#stream#name</code></td>
<td><code>#l</code></td>
</tr>
<tr>
<td><code>##parameter</code></td>
<td><code>#p</code></td>
</tr>
<tr>
<td><code>##profile</code></td>
<td><code>#P</code></td>
</tr>
<tr>
<td><code>##resource#group</code></td>
<td><code>#r</code></td>
</tr>
<tr>
<td><code>##value</code></td>
<td><code>#v</code></td>
</tr>
</tbody>
</table>
Localizing Game Projects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create multiple-language, region-specific versions of your game using Lumberyard's localization system.

The localization system supports Unicode keyboard input support and language-specific loading of font assets.

With this system, you can do the following:

- Easily modify localization assets and definitions. These are stored in XML files.
- Load and unload localized text assets as you need them. Localized text assets are grouped with tags that you choose.
- Reference localized text using the UI system's text component.
- Use the UI Editor to rapidly preview your UI in multiple languages.
- Customize the font texture configuration. This makes it possible for the engine to render a large number of unique characters or glyphs.

Configuring and Loading Localization Data

Lumberyard has two main types of localization XML assets:

- Tag definition XML (p. 2415)
- Localized text XML (p. 2416)

After you prepare your assets and place them in the preferred location, see the Localization System (p. 2417) documentation to ensure that your game is ready to use your assets.

English is the default language for a game project. To change languages, see Console Variables (p. 2418).

Tag Definition XML

The Tag Definition XML file groups localized text XML files by tag name. It must be loaded in C++ at runtime (p. 2418), typically at startup.

The following is an example of how the tag definition XML file appears:

```
<localization>
  <init>
    <entry>text_ui_menus.xml</entry>
  </init>
  <singleplayer>
    <entry>text_ui_sp_dialog.xml</entry>
  </singleplayer>
  <multiplayer>
    <entry>text_ui_mp_dialog.xml</entry>
  </multiplayer>
</localization>
```
This file should reside within your game project directory. For example, in the Samples Project, this file is located at dev/SamplesProject/libs/Localization/localization.xml. However, its specific location is inconsequential as long as the localization manager can load it at runtime. For more information, see Localization Initialization (p. 2417).

Localized text XML

Localized text XML files contain the text resources for a specific language. This way, your game project can reference the localization keys. This causes the localization system to return the corresponding value for the currently configured language. For more information, see Localizing Text (p. 2418).

You can edit these files manually, in Microsoft Excel, or in most industry-standard CAT tools.

For an example of an English localized text XML, open dev/SamplesProject/Localization/English_xml/text_ui_menus.xml.

For an example of the same localized text XML in Korean, open dev/SamplesProject/Localization/Korean_xml/text_ui_menus.xml.

Localization Checklist

To use localization in your game, verify that you have done the following:

1. Created a tag definition XML (p. 2415) and placed it within your game project directory.
2. Created localized text XML (p. 2416) assets for at least one language, also located within your game project directory.
3. Configured the g_language and sys_localization_folder console variables (p. 2418) appropriately.
   • g_language defaults to English.
   • sys_localization_folder must point to a game project sub-directory where you store your localized text XML assets in language-specific sub-directories.
4. Initialized the Localization Manager (p. 2418) in your game's startup code.

After completing these steps, you can start localizing text by passing in the localized key string to the localization manager. Note that some Lumberyard systems, such as the UI System (p. 2416), already attempt to localize text for you (p. 2418).

UI System Localization Features

By default, all Text (p. 2943) component values are passed to the Localization Manager (p. 2418) for localizing. If the value of a Text (p. 2943) component matches a localization key for the currently loaded language, then the localized text is rendered.

For an example, open the Samples Project (p. 146). In the UI Editor (p. 2874), load the following UI canvas: dev/Gems/LyShineExamples/Assets/UI/Canvases/LyShineExamples/Localization/Text.uicanvas.

You can quickly test a UI canvas that contains localized text. To change the current language in the UI Editor (p. 2874), choose View, Set Current Language.
Font Features

Language-Specific Font Assets

You can group Font Family XML (p. 3000) definitions by using a lang attribute. This attribute tells the localization system to load font assets only for the currently actively language. To see an example, open the Samples Project (p. 146) and, in the UI Editor (p. 2874), open dev/Gems/LyShineExamples/Assets/UI/Fonts/LyShineExamples/NotoSans/NotoSans.fontfamily.

Unique Characters

Some languages feature a large variety of unique characters. Font XML files contain attributes for adjusting font texture resolution and number of slots for storing unique characters. For more information, see Configuring Font Quality (p. 3002).

For an example of unique character support in action, open the Samples Project (p. 146), then load the UiFeatures level. Press Ctrl+G to play the game. Choose Language Support, then choose either Unique Characters: 32px or Unique Characters: 64px.

Localization Initialization

Initializing the localization system requires the following:
Localization Initialization

- C++ call to initialize the Localization Manager at game startup
- Tag Definition XML
- Localized text XML

Localization Manager

At game startup, you must initialize the Localization Manager (ILocalizationManager) with the tag definition XML.

For example, the following code snippet is from dev/SamplesProject/Gem/Code/Source/Core/GameplaySampleGame.cpp within the Samples Project:

```cpp
void LoadLocalizationData()
{
    ILocalizationManager* pLocMan = GetISystem()->GetLocalizationManager();
    if (pLocMan)
    {
        AZStd::string localizationXml("libs/localization/localization.xml");
        if (pLocMan->InitLocalizationData(localizationXml.c_str()))
        {
            const bool initLocSuccess = pLocMan->LoadLocalizationDataByTag("init");
            AZ_Error("Localization", initLocSuccess, "LoadLocalizationData() failed to load localization file=%s", localizationXml.c_str());
        }
    }
}
```

The code sample above does the following:

- Initializes the localization manager with the tag definition XML.
- Loads all the localized text XML for the current language that is grouped by the init tag within the tag definition XML.

Console Variables

The following two console variables, or CVars, affect the localization system:

- **g_language**
  The name of the current language that the game uses. Is set to *English* by default.

- **sys_localization_folder**
  The directory used for finding localization assets. This directory contains either language PAK files or 'loose' localized text XML files.

For example, in the Samples Project, the game config file, /dev/SamplesProject/game.cfg, sets sys_localization_folder to Localization. In the game directory structure, this defines the location as dev/SamplesProject/Localization.

Localizing Text

After you properly initialize the localization manager, you can localize text to the current language. To do this, you pass in the localization key that you want to localize.

For example:
string locText;
gEnv->pSystem->GetLocalizationManager()->LocalizeString("@ui_Hello", locText);

Assuming the key @ui_Hello is localized for the current language, the contents of locText will contain the Unicode (UTF8) encoding for the localized value contained within the localized text XML.

**UiTextComponent**

Lumberyard's UI System (p. 2874), contained in the LyShine Gem and installed by default, features a text (p. 2943) component with built-in support for automatically localizing text for you. When you enter a valid localized text key (such as @ui_Hello as shown in the previous example (p. 2418)), the engine replaces the contents of the rendered string with the localized text. That key must be localized for the current language.

To see the source code for examples of calls to the localization manager, open ui/Gems/LyShine/Code/Source.UiTextComponent.cpp.
Scripting in Amazon Lumberyard

Lumberyard includes two scripting technologies for automating gameplay: Script Canvas and Lua.

**Script Canvas**

A visual scripting environment that you can use to create game logic and behaviors without writing code. In the Script Canvas editor, you create, connect, and rearrange graphical nodes that provide a visual representation of the logic flow.

For more information, see Creating Gameplay with Script Canvas (p. 2420).

**Lua**

A powerful, fast, lightweight, embeddable scripting language. Lua facilitates quick iteration in your game project because you can run your changes immediately without needing to recompile your source code.

For more information, see Writing Lua Scripts (p. 2683).

Lumberyard's functionality is exposed to Script Canvas and Lua by the behavior context. For more information, see Behavior Context (p. 1053).

**Topics**

- Creating Gameplay with Script Canvas (p. 2420)
- Writing Lua Scripts (p. 2683)
- Using Script Events (p. 2762)

Creating Gameplay with Script Canvas

Script Canvas is a visual scripting environment for Amazon Lumberyard that you can use to create game logic and behaviors. Integrated with Lumberyard's component entity system, Script Canvas offers an approachable and easy-to-read environment to author behaviors using the same framework as Lua and C++. With its visual scripting options, you can use Script Canvas to create scripts for game logic without needing to know how to code. Event-driven scripts let you produce small scripts that don't require complex logic to maintain states.

Script Canvas is also an extensible system. You can build your own reusable Script Canvas functions and, since Script Canvas is based on AzCore, you can create custom Script Canvas nodes that use the serialization, reflection, modularization, and EBus messaging systems.
To enable Script Canvas for Lumberyard, you must enable the Script Canvas (p. 1201) gem and its dependencies.

Topics
- Learning Guide (p. 2421)
- Common Tasks (p. 2468)
- Editor Reference (p. 2479)
- Programmer Guide (p. 2543)
- Script Canvas Node Reference (p. 2567)

Learning Guide

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In this Learning Guide, we provide you with a tour of the Script Canvas interface, several tutorials to get you started using the visual scripting environment, and a guide to additional resources for learning Script Canvas. Additionally, there's a glossary of terms you can continue referring back to as you learn your way around.

Topics
- Script Canvas Editor Interface (p. 2421)
- Script Canvas Concepts and Terms (p. 2423)
- Script Canvas Tutorials (p. 2424)
- More Learning Opportunities with Script Canvas (p. 2466)

Script Canvas Editor Interface

You can open the Script Canvas editor from Lumberyard Editor.

To open the Script Canvas editor
1. In Lumberyard Editor, choose Tools, Script Canvas.
2. Choose File, New Script or drag a node from the Node Palette and drop it on the canvas.
In the **Script Canvas** editor, you can do the following:

A. Use the menu bar to do the following:
   - Create, save, and open your scripts.
   - Cut, copy, or undo actions.
   - Change the **Script Canvas** editor view.

B. Use the tabs to switch between scripts.

C. In the **Node Palette**, you can search for nodes.

D. You can drag a node from the **Node Palette** to the canvas or right-click the canvas for the menu to appear.

E. On a node, you can specify values for the parameters.

F. Drag to connect the input pin of a node to an output pin of another node. This line creates a connection between the nodes.

G. The **Variable Manager** shows the variables that are used in the script. You can add or delete variables and set their default values. To create **Get**, **Set**, or **OnValueChanged** nodes, you can drag a variable from the **Variable Manager** onto the script.

H. In the **Node Inspector**, you can view and modify the properties for a selected node.

### Additional Tools

**Script Canvas** editor has the following additional menus and tools.

**Bookmarks**

View and modify your saved bookmarks. You can use bookmarks to save locations on your script and then use keyboard shortcuts to move to that location. For more information, see Adding Bookmarks for Script Canvas (p. 2534).

**Comments**

You can create floating blocks of text on the Script Canvas editor canvas. For more information, see Commenting Nodes (p. 2524).
Script Canvas Concepts and Terms

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following concepts and terms are used in **Script Canvas**:

**Script**

A script is a collection of nodes, node properties, and node connections that, when combined, create a visual script.

**Node**

Nodes represent the data, events, and actions that you use to create logic and behaviors in **Script Canvas**.

See the following node types:

**Event nodes**

Event nodes subscribe to event bus (EBus) handlers to listen for events to occur. Examples include entering a trigger area, colliding with an object, turning off a light, and when the game ticks.

For more information about using the EBus interface, see [Working with the Event Bus (EBus) system](p. 1851).

**Action nodes**

Action nodes are used to get or send data across an EBus. Examples of action nodes include getting the mass of an entity, turning on a light, setting the text of a UI element, and playing an animation.

For more information about using the EBus interface, see [Working with the Event Bus (EBus) system](p. 1851).

**Variable and data nodes**

Variable and data nodes represent the custom data that can be required to build game logic. You can use these nodes to make counters, store entity references, specify a direction, define a color, and so on. Variable nodes are added to a script to declare and initialize them. Use get and set nodes to retrieve or set the variable's value.

The following are the commonly used data types in **Script Canvas**:

- Boolean
- Color
- Entity
- Number
- String
- Transform
- Vector 2/3/4

**Logic**

Logic nodes include comparison and timing operations. You can use logic nodes to check whether two values are equal, control the execution of nodes, delay the execution of a node for a specific amount of time, and more.
Math

Math nodes enable math operations, such as arithmetic, geometry, algebra, calculus and so on.

Debugging

Debugging nodes verify whether a script is functioning as expected. You can use debugging nodes to print data to the console or viewport and check for errors. These nodes pass logic flow, but do not execute in release builds.

User defined

You can build your own nodes for your project’s specific needs. For more information, see Creating Custom Nodes in Script Canvas (p. 2553).

Node Palette

The Node Palette contains a searchable list of nodes. By default, the palette is docked to the left of the Script Canvas editor.

To display the Node Palette

1. From Lumberyard Editor, choose Tools, Script Canvas.
2. In the Script Canvas window, do one of the following:
   - Choose Tools, Node Palette.
   - Press Ctrl+Shift+L.

   Note
   If you have an existing script open, you can right-click the canvas for the menu to appear.

Node Inspector

The Node Inspector shows the properties of the node. You can edit each property in the inspector or directly in the node. By default, this window doesn’t appear in the editor.

To display the Node Inspector

1. Open Script Canvas (p. 2421).
2. Do one of the following:
   - Click Tools, Node Inspector.
   - Press Ctrl+Shift+I.

Script Canvas Tutorials

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following Script Canvas tutorials to learn about the Lumberyard visual scripting system.

Prerequisites

- Set the Samples Project (p. 146) as the default project. For more information, see Choosing a Game Project to Open (p. 47).
• If your **Node Palette** is missing nodes, you can update your preferences to show hidden nodes. For more information, see *Script Canvas Node Reference* (p. 2567).

• To access diagnostic nodes, including the **Print** node, you must enable the **Script Canvas Diagnostic Library** gem and then build your game project. For more information, see *Enabling Gems* (p. 1064).

**Topics**

- *Script Canvas Tutorial: Creating a Controllable Entity* (p. 2425)
- *Script Canvas Tutorial: Opening and Closing a Door with Trigger Areas and Variables* (p. 2435)
- *Script Canvas Tutorial: Shooting a Target by Spawning Entities and Detecting Collisions* (p. 2446)

**Script Canvas Tutorial: Creating a Controllable Entity**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

This tutorial walks you through the steps to create a simple sphere that you can control with keyboard input.

**Example**

![Sphere](#)

In addition to node basics, input, movement, and logging, you learn the following key concepts:

• Adding nodes
• Creating execution and data connections
• Adding event nodes
• Adding action nodes
Prerequisites

- Set the Samples Project (p. 146) as the default project. For more information, see Choosing a Game Project to Open (p. 47).
- If your Node Palette is missing nodes, you can update your preferences to show hidden nodes. For more information, see Script Canvas Node Reference (p. 2567).
- To access diagnostic nodes, including the Print node, you must enable the Script Canvas Diagnostic Library gem and then build your game project. For more information, see Enabling Gems (p. 1064).

Topics

- Step 1: Create a Level (p. 2426)
- Step 2: Create an Entity and Add Components to Create a Controllable Entity (p. 2426)
- Step 3: Create an Input Script with Script Canvas (p. 2429)
- Step 4: Assign the Script to Your Entity and Test the Script (p. 2431)
- Step 5: Create a Script to Move Your Sphere (p. 2431)
- Step 6: Add Movement on the X-Axis for Your Sphere (p. 2432)

Step 1: Create a Level

Before you create a sphere that you can control with keyboard input, you need to create a level.

To create a level

1. In Lumberyard Editor, choose File, New.
2. In the New Level dialog box, enter firstscriptcanvas and then click OK.
3. In the Generate Terrain Texture dialog box, choose 512x512 and then click OK.

Step 2: Create an Entity and Add Components to Create a Controllable Entity

To create a sphere that you can control with keyboard input, you create an entity and add the Mesh (p. 684), Rigid Body component, Mesh Collider, and Input (p. 642) components. You also create an input mapping that converts WASD keyboard input into movement in the x and y directions.

To create a controllable entity

1. In Lumberyard Editor, right-click in the Perspective viewport and choose Create entity.
2. In the Entity Inspector, do the following:
   a. For Name, enter Player.
   b. Click Add Component and then choose the Mesh component.
   c. In the Mesh component, for Mesh asset, click the browse (...) button, select the \SamplesProject\Objects\default\primitive_sphere.cfg file and then click OK.
3. In the viewport, use the Move tool (p. 201) to select the z-axis and move the entity off the ground.

4. In the Entity Inspector, click Add Component and then add the following components:
   - Rigid Body Physics
   - Mesh Collider
   - Input

5. In the Input component, click the Input Bindings Editor icon. This opens the Asset Editor.

6. In the Asset Editor, choose File, New, Input Bindings and for File name, enter player and then click Save.

7. In the Edit Asset window, do the following:
   a. For Input Event Groups, click + to add a new input event group.
   b. Expand the input event group. For Event Name, enter move_x.
   c. For Event Generators, click + to add an event generator.
   d. In the Class to create dialog box, click OK to add an input class.
   e. Expand move_x, Event Generators, gamepad_button_a. For Input Device Type, select keyboard.
   f. For Input Name, select keyboard_key_alphanumeric_A.
   g. For Event value multiplier, enter -1.
   h. Repeat steps C – E. For Input Name, select keyboard_key_alphanumeric_D. For Event value multiplier, use the default value of 1.
   i. Expand the keyboard_key_alphanumeric_A event generator and verify that your settings appear like the following.
8. In the **Edit Asset** window, do the following:

   a. For **Input Event Groups**, click + to add a new input event group.

   b. Expand the input event group. For **Event Name**, enter *move_y*.

   c. For **Event Generators**, click + to add an event generator.

   d. In the **Class to create** dialog box, click OK to add an input class.

   e. Expand *move_y*, **Event Generators**, *gamepad_button_a*. For **Input Device Type**, select **keyboard**.

   f. For **Input Name**, select **keyboard_key_alphanumeric_W**. For **Event value multiplier**, use the default value of 1.

   g. Repeat steps C – E. For **Input Name**, select **keyboard_key_alphanumeric_S**. For **Event value multiplier**, enter -1.

   h. Expand **keyboard_key_alphanumeric_W** and verify that your settings appear like the following.
Step 3: Create an Input Script with Script Canvas

Now that you set up a sphere with physics and input mapping, you can create your first script with the Script Canvas editor.

To create an input script
1. In Lumberyard Editor, choose Tools, Script Canvas.
2. In the Script Canvas editor, choose File, New Script.
3. After the graph loads, choose File, Save As.
4. In the **Save As** dialog box, for **File name**, enter `player` and then click **Save**.

5. In the **Node Palette**, enter **input** in the search box.

   **Note**
   You can also access the list of nodes by right-clicking in an empty area on the graph.

6. Under **Gameplay, Input**, drag **Input Handler** from the **Node Palette** to the graph. **Input Handler** is an event node. When an event occurs, the event node sends a message to the graph.

7. For **Event Name**, enter `move_y`. This tells the node to listen for the input event.

![Input Handler](image)

8. In the **Node Palette**, enter **Print** in the search box.

9. Under **Utilities, Debug**, drag **Print** from the **Node Palette** to the graph. **Print** is an action node. When you execute an action node, it completes actions such as request data, set data, manipulate data, and trigger functions. Action nodes also print data to the editor console, allowing you to check values as the script is being executed.

10. From **Input Handler**, drag the **Held** pin to connect it to the **In** pin of the **Print** node. This connection tells the **Print** node to execute after the input handler receives a held event.

11. From **Input Handler**, drag the **Value** pin to connect it to the **Value** pin of the **Print** node. This connection tells the **Input Handler** node to pass the input event value from the input handler to the first argument on the **Print** node.

   **Note**
   - Node execution always flows from the left side to the right side of a node.
   - Data is always input on the left side of a node and output from the right side of the node.

   For more information, see **Inputs, Outputs, and Connection Types** (p. 2497).

12. Verify that your `player.scriptcanvas` file looks like the following.

![Print](image)

13. In the **Script Canvas** editor, choose **File, Save** or press **Ctrl+S**.
Step 4: Assign the Script to Your Entity and Test the Script

The script that you made outputs the value of the input event. Now you can assign the script to your entity and test the script.

Note
The tutorial's keyboard input doesn't work in a remote desktop session.

To assign and test your script

1. In Lumberyard Editor, select the Player entity that you created.
2. In the Entity Inspector, click Add Component, and then choose the Script Canvas component.
3. Under Script Canvas, click the browse (...) button, select the player.scriptcanvas file, and then click OK.
4. Press ` to open the console window or choose Tools, Console.
5. Press Ctrl+G to enter game mode.
6. To trigger your input event, press W and then press S. The Console pane in Lumberyard Editor outputs the value of the input event when the move_y event is triggered. In this case, the Console pane outputs 1 when you press W and outputs -1 when you press S.

7. When you are done testing your script, press Esc.

Step 5: Create a Script to Move Your Sphere

Now that you've successfully created your first script, you can add nodes to move your sphere. You also modify the input event value to control the movement speed.

To move your sphere

1. In the Script Canvas editor, in the Node Palette, enter multiply in the search box.
2. Under Math, drag Multiply from the Node Palette to the graph.
3. From Input Handler, drag the Held pin to connect it to the In pin of the Multiply node. This connection executes the Multiply node after the input handler receives a held event.
4. From Input Handler, drag the Value pin to connect it to the Value A pin of the Multiply node.
5. In the **Multiply** node, for **Value B**, enter **0.1**. This smaller value for the input event throttles the movement speed.

6. In the **Node Palette**, enter **move** in the search box.

7. Under **Entity, Transform**, drag **Move Entity** from the **Node Palette** to the graph.

8. In the **Node Palette**, enter **create from values** in the search box.

9. Under **Math, Vector3**, drag **Create From Values** from the **Node Palette** to the graph. You use this node to build a vector 3 from a number.

10. From **Multiply**, drag the **Out** pin to connect it to the **In** pin of the **Create From Values** node.

11. From **Multiply**, drag the **Result** pin to connect it to the **Y** pin of the **Create From Values** node. You only need a y direction, so use the default value (0) for **X** and **Z**.

12. From **Create From Values**, drag the **Out** pin to connect it to the **In** pin of the **Move Entity** node.

13. From **Create From Values**, drag the **Vector 3** pin to connect it to the **Direction** pin of the **Move Entity** node.

14. Verify that your **player.scriptcanvas** graph looks like the following.

15. Save your graph.

16. In Lumberyard Editor, press **Ctrl+G** to enter game mode and test your script.

17. To move the sphere forward, press **W**. To move the sphere backwards, press **S**.

18. When you are done testing your script, press **Esc**.

**Step 6: Add Movement on the X-Axis for Your Sphere**

Now that you've converted your input event value to a direction that moves the entity on the y-axis, you can add movement on the x-axis.

**To add movement on the x-axis**

1. In the **Script Canvas** editor, drag on the graph to select the **Input Handler, Multiply, Create From Values**, and **Move Entity** nodes. An orange outline appears around the selected nodes.
2. Press **Ctrl+C** to copy the selected nodes.
3. Press **Ctrl+V** to paste the copied nodes and their connections. This allows you to clone the existing script and modify the cloned version with the appropriate settings for movement on the x-axis.
4. While still selected, move the nodes so they don't overlap the copied nodes.
5. In the duplicate **Input Handler** node, for **Event Name**, enter `move_x`.

**Example**

Your *player.scriptcanvas* graph should look like the following.

6. For the duplicate **Multiply** node, move the connection from the Y pin to the X pin on the duplicate **Create From Values** node.

**Note**

To disconnect a connection, drag it to an empty part of the canvas. To delete a connection, press **Alt** and click the connection.
7. Verify that your `player.scriptcanvas` file graph looks like the following.

Example

8. Save your graph.

9. In Lumberyard Editor, press **Ctrl+G** to enter game mode and test your script.

10. Do the following:
    
    a. To move the sphere forward, press **W**.
    b. To move the sphere backwards, press **S**.
    c. To move the sphere from side to side, press **A** and **D**.

11. When you are done testing your script, press **Esc**.
Script Canvas Tutorial: Opening and Closing a Door with Trigger Areas and Variables

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In the following tutorial, you create a door that opens and closes when your controllable sphere enters and exits a trigger area.

Example

This involves several tasks:

- Create a trigger area
- Add event nodes
• Add variables

Prerequisites

• Complete the Script Canvas Tutorial: Creating a Controllable Entity (p. 2425).

Topics

• Step 1: Create a Door and Trigger Area (p. 2436)
• Step 2: Create a Script to Open and Close the Door (p. 2441)

Step 1: Create a Door and Trigger Area

To create a door and trigger area, create an entity with child entities and then add components to the child entities.

To create a door and trigger area

1. In Lumberyard Editor, right-click the Perspective viewport near your controllable sphere and choose Create entity.

2. In the Entity Inspector, for Name, enter Door Group.

3. Do the following to create a door:
   a. In the Entity Outliner, right-click Door Group and choose Create child entity. This child entity is your door.
   b. In the Entity Inspector, for Name, enter Door Mesh.
c. For the **Door Mesh** entity, click **Add Component** and then choose the **Mesh** (p. 684) component.

d. In the **Mesh** component, for **Mesh asset**, click the browse (…) icon and select the `SamplesProject\Objects\Primitives\box_1x1.cgf` file.

e. Click **Add Component** and then choose the **Static Physics** component.

f. Click **Add Component** and then choose the **Mesh Collider** component. This component defines the collision shape for the **Door Mesh** entity.

g. In the **Transform** (p. 878) component, for **Scale**, set X to **2.5**, Y to **0.5**, and Z to **4.0**.

h. Verify that your **Door Mesh** entity looks like the following.
Example

4. Do the following to create a trigger area:
   a. In the **Entity Outliner**, right-click **Door Group** and choose **Create child entity**. This child entity is your trigger area.
   b. In the **Entity Inspector**, for **Name**, enter **Door Trigger**.
   c. Click **Add Component** and then choose the **Trigger Area (p. 890)** component.
d. With the Move tool (p. 201), adjust the Door Trigger entity so that its Z position fits the Door Mesh entity.

e. In the Trigger Area component, click Add Required Component and choose Box Shape.

f. For the Box Shape component, for Dimensions, set X to 3.0, Y to 9.0, and Z to 6.0.

g. Click Add Component and then choose the Script Canvas (p. 818) component.

h. In Lumberyard Editor, choose Tools, Script Canvas.

i. In the Script Canvas editor, choose File, New Script.

j. After the new canvas loads, choose File, Save As.

k. For File name, enter door and then click Save.

l. In the Script Canvas component, click the browse (...) icon, select the door.scriptcanvas file, and then click OK.

5. Verify that your Door Trigger entity looks like the following.
Step 2: Create a Script to Open and Close the Door

Now that you've set up your door and trigger area, you can create a script that opens and closes the door when another entity enters or leaves the trigger area.

To create a script that opens and closes the door

1. In the Script Canvas editor, open the door.scriptcanvas file.
2. In the Node Palette, enter trigger in the search box and under Gameplay, Trigger Area, drag On Area Entered to the canvas.
   
   Note
   
   With this event node you can easily use the Lumberyard EBus messaging system. For more information, see Working with the Event Bus (EBus) system (p. 1851).

3. In the Trigger Area node, click Add/Remove Events and then select the On Area Exited check box. This exposes the entered and exited events from the Trigger Area EBus.

4. In the bottom-right pane, in the Variable Manager, click Create Variable.
   
   Note
   
   You can store and modify persistent values in your graph with variable nodes. For more information, see Managing Script Canvas Variables (p. 2482).

5. Select Vector3, double-click Variable 1 and then rename the variable to opened_position.
6. In the Node Inspector, specify 2 for the Z value. A positive value for the z-axis slides the door up. Because the door entity is a child of Door Group, you can specify local relative positions to control the open and closed positions of the door.
7. Create another Vector3 variable and name it closed_position. Keep the default value of 0, 0, 0.
8. Create another Vector3 variable and name it current_position. Keep the default value of 0, 0, 0. This variable sets the door's current position.
9. Create another Vector3 variable and name it `destination_position`. Keep the default value of 0, 0, 0. This variable sets the door’s destination position when the entity enters and exits the trigger area.

**Example**

You should have four Vector3 variable nodes such as the following.

![Variable Manager](image)

10. In the **Variable Manager**, do the following:

   a. Select and drag the `open_position` node to the canvas and then click **Get opened_position**.
   b. Select and drag the `closed_position` node to the canvas and then click **Get closed_position**.
   c. Select and drag two `destination_position` nodes to the canvas and then click **Set destination_position**.

11. In the canvas, make the following connections:

   a. From **On Area Entered**, drag the Out pin to connect it to the In pin of a **Get opened_position** node.
   b. From **On Area Exited**, drag the Out pin to connect it to the In pin of the **Get closed_position** node.
   c. From **Get opened_position**, drag the Out pin to connect it to the In pin of the **Set destination_position** node.
   d. From **Get opened_position**, drag the Vector3 pin to connect it to the Vector3 pin of the **Set destination_position** node.
   e. From **Get closed_position**, drag the Out pin to connect it to the In pin of the other **Set destination_position** node.
   f. From **Get closed_position**, drag the Vector3 pin to connect it to the Vector3 pin of the other **Set destination_position** node.
   g. Verify that your `door.scriptcanvas` graph looks like the following.
Example

12. Do the following to get the Door Mesh entity's position and interpolate to the destination:

   a. In the Node Palette, enter `get local` in the search box.

   b. Under Entity, Transform, drag Get Local Translation to the canvas. You can use this node to get the current position of the Door Mesh entity and interpolate to the destination. A local translation applies to the translation of the entity relative to its parent.

   c. In the Get Local Translation node, pause on the Source text box and click the target button. When selected, the target button has an orange outline.

   d. In the Entity Outliner, select Door Mesh to assign the Door Mesh entity to the Source property in the Get Local Translation node.
Note
To reset an entity reference, right-click twice on the Source text box and choose Set to Self.

e. In the Script Canvas editor, from both Set destination_position nodes, drag the Out pins to connect it to the In pin for Get Local Translation.

Note
When multiple connections enter a single logic pin, the node is executed each time either execution is triggered. The node is executed more than once in the same game tick if multiple executions are triggered simultaneously.

13. Do the following to execute nodes for a specified amount of time, in seconds:

   a. In the Variable Manager, drag current_position to the canvas and click Set current_position.

   b. From Get Local Translation, drag the Out pin to connect it to the In pin of the Set current_position node.

   c. From Get Local Translation, drag the Translation pin to connect it to the Vector3 pin of the Set current_position node.

   d. In the Node Palette, enter duration in the search box and under Timing, drag Duration to the canvas.

   e. From Set current_position, drag the Out pin to connect it to the Start pin of the Duration node. Triggering the Duration node resets the time.

   f. In the Duration node, for Duration, enter 1.0 (seconds).
Example

14. Do the following to set up interpolation between the current position and the destination:

   a. In the Variable Manager, select and drag `current_position` to the canvas and then click Get `current_position`.

   b. In the Variable Manager, select and drag `destination_position` to the canvas and then click Get `destination_position`.

   c. In the Node Palette, enter `lerp` in the search box and under Math, Vector3, drag Lerp from the Node Palette to the canvas. This node blends two values based on the Percentage property.

   d. From Duration, drag the Out pin to connect it to the In pin of the Get `current_position` node.

   e. From Duration, drag the Elapsed pin to connect it to the Percentage pin of the Lerp node.

   f. From Get `current_position`, drag the Out pin to connect it to the In pin of the Get `destination_position` node.

   g. From Get `current_position`, drag the Vector3 pin to connect it to the Start pin of the Lerp node.

   h. From Get `destination_position`, drag the Out pin to connect it to the In pin of the Lerp node.

   i. From Get `destination_position`, drag the Vector3 pin to connect it to the End pin of the Lerp node.

Example
15. Do the following to set the position of the door when the **Duration** node blends between the current and destination positions:

   a. In the **Node Palette**, enter **set local translation** in the search box and under **Entity, Transform**, drag **Set Local Translation** to the canvas.

   b. From **Lerp**, drag the **Out** pin to connect it to the **In** pin of the **Set Local Translation** node.

   c. From **Lerp**, drag the **Vector3** pin to connect it to the **Translation** pin of the **Set Local Translation** node.

   d. In the **Set Local Translation** node, pause on the **Source** text box and click the target button.

   e. In the **Entity Outliner**, select **Door Mesh** to assign the **Door Mesh** entity to the **Source** property in the **Set Local Translation** node.

   f. Verify that your **door.scriptcanvas** graph looks like the following.

   **Example**

   ![Diagram of door scriptcanvas graph]

16. Save your graph.

17. In Lumberyard Editor, press **Ctrl+G** to enter game mode and test your script.

18. To move the sphere forward into the door trigger area and slide open the door, press the **W, A, D** keys.

19. To move the sphere backwards out of the trigger area and slide the door closed, press **S**.

20. When you are done testing your script, press **Esc**.

**Script Canvas Tutorial: Shooting a Target by Spawning Entities and Detecting Collisions**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://www.aws.amazon.com/about-aws/whats-new/) or visit the [AWS Game Tech blog](https://aws.amazon.com/en_game/tech/) to learn more.

This tutorial builds on what you learned in the previous two tutorials. In the following tutorial, you create a controllable sphere that shoots projectiles at a target.
Example

This involves several tasks:

- Spawn entities
- Add scripts that run on the spawned entities
- Set up collision
- Add tag filtering

Prerequisites

- Complete the Script Canvas Tutorial: Creating a Controllable Entity (p. 2425) and Script Canvas Tutorial: Opening and Closing a Door with Trigger Areas and Variables (p. 2435).

Topics

- Step 1: Set Up a Projectile Entity (p. 2448)
- Step 2: Create a Script to Propel the Projectile Forward (p. 2452)
- Step 3: Save the Projectile Entity as a Slice (p. 2454)
- Step 4: Set Up the Player Entity to Spawn the Slice (p. 2456)
- Step 5: Set Up the Input Binding for the Projectile (p. 2457)
- Step 6: Add Logic to the Player Script to Spawn the Projectile (p. 2458)
- Step 7: Create a Target Entity (p. 2460)
- Step 8: Add Logic to the Projectile to Destroy Target Entities (p. 2463)
Step 1: Set Up a Projectile Entity

To create a projectile entity, create an entity and then add the Mesh (p. 684), Rigid Body component, Mesh Collider, and Script Canvas (p. 818) components.

To create a projectile entity

1. In Lumberyard Editor, right-click in the Perspective viewport near your sphere and choose Create entity.
2. In the Entity Inspector, do the following:
   a. For Name, enter Projectile.
   b. In the Transform component, for Scale, set X to 0.25, Y to 0.25, and Z to 0.25.
   c. Click Add Component and then choose the Mesh component.
   d. In the Mesh component, for Mesh asset, click the browse (...) button, select the \SamplesProject\Objects\default\primitive_sphere.cgf file, and then click OK.
3. In the **Entity Inspector**, click **Add Component** and then choose the following components:
   - Rigid Body Physics
   - Mesh Collider
   - Script Canvas

4. In Lumberyard Editor, choose **Tools, Script Canvas**.

5. In the **Script Canvas** editor, choose **File, New Script**.

6. After the graph loads, choose **File, Save As**.

7. In the **Save As** dialog box, for **File name**, enter `projectile` and then click **Save**.

8. In the **Script Canvas** component, for **Script Canvas Asset**, click the browse (...) button, select the `projectile.scriptcanvas` file, and then click **OK**.
9. Verify that the **Projectile** entity looks like the following.
Example

![Image of Lumberyard User Guide](image-url)
Step 2: Create a Script to Propel the Projectile Forward

Now that you’ve set up a projectile, you can create a script to propel the projectile forward when it spawns.

To create a projectile script

1. In the Script Canvas editor, open the projectile.scriptcanvas file, if it’s not still open from Step 1: Set Up a Projectile Entity (p. 2448).
2. In the Node Palette, enter start in the search box.
3. Under Utilities, drag On Graph Start from the Node Palette to the graph. This event node executes only for the first tick after the entity has initialized.
4. In the Node Palette, enter Get Forward.
5. From Entity, Transform, drag the Get Forward node to the graph.
6. From On Graph Start, drag the Out pin to connect it to the In pin of the Get Forward node.

7. In the Get Forward node, for Scale, enter 200.
8. In the Node Palette, enter Set Velocity and drag the node to the graph.
9. From Get Forward, drag the Out pin to connect it to the In pin of the Set Velocity node.
10. From Get Forward, drag the Forward pin to the Velocity pin of the Set Velocity node.

11. To set a lifetime for the projectile, do the following:
a. In the **Node Palette**, enter **delay** in the search box.

![Delay node]

b. Under **Timing**, drag **Delay** from the **Node Palette** to the graph.

c. From **On Graph Start**, drag the **Out** pin to connect it to the **In** pin of the **Delay** node.

d. In the **Delay** node, for **Time**, enter **1.0**.

12. To destroy the projectile after the set time, do the following:

a. In the **Node Palette**, enter **destroy** in the search box.

b. Under **Entity, Game Entity**, drag **Destroy Game Entity and Descendants** from the **Node Palette** to the graph.

![Destroy Game Entity And Descendants node]

c. From **Delay**, drag the **Out** pin to connect it to the **In** pin of the **Destroy Game Entity and Descendants** node.
d. Verify that your projectile.scriptcanvas graph looks like the following.

Example

13. In the Script Canvas editor, choose File, Save. You can also press Ctrl+S.

Step 3: Save the Projectile Entity as a Slice

Now that you've created a projectile script, you can save the projectile entity as a slice.

To save the projectile entity as a slice

1. In Lumberyard Editor, in the Entity Outliner, right-click Projectile and choose Create slice.
2. In the Save As dialog box, for File name, enter projectile and then click Save.
3. To enable the projectile to spawn, do the following:
   a. In the Asset Browser, navigate to the projectile.slice file that you just created.
b. Right-click `projectile.slice` and choose **Set Dynamic Slice**. Scripts that run on a dynamic slice will properly remap source entity assignments to the entity that the graph runs on.

4. In the **Entity Outliner**, right-click `Projectile` and then click **Delete**. Because you will spawn the projectile entity dynamically from your player script, you no longer need the entity in your scene.
Step 4: Set Up the Player Entity to Spawn the Slice

Now that you've created a slice, you can set up your Player entity to spawn the slice.

To set up the player entity to spawn the slice

1. In Lumberyard Editor, in the Entity Outliner, right-click the Player entity and choose Create child entity.
2. Enter a name for the child entity such as spawner.
3. In the Transform component, for the Translate Y property, enter 1.0. This sets the spawn point 1 meter in front of the parent entity.
4. Select the child entity and in the Entity Inspector, click Add Component, and then choose the Spawner (p. 859) component.
5. In the Spawner component, for Dynamic slice, click the browse (...) icon, select the projectile.slice file, and then click OK.

6. Verify that the spawner entity looks like the following.
Step 5: Set Up the Input Binding for the Projectile

Now that you've added the Spawner component, you can set up the input binding to shoot the projectile when you press the Spacebar.

To set up the projectile input binding

1. In Lumberyard Editor, in the Entity Outliner, select the Player entity.
2. In the Entity Inspector, under Input, click the Input Bindings Editor icon.
3. In the Edit Asset window, do the following:
   a. For Input Event Groups, click + to add a new input event group.
   b. Expand the input event group. For Event Name, enter shoot.
   c. For Event Generators, click + to add an event generator.
   d. In the Class to create dialog box, click OK to add an input class.
   e. Expand shoot, Event Generators, gamepad_button_a. For Input Device Type, choose keyboard.
f. For **Input Name**, choose `keyboard_key_edit_space`.

g. Verify that your settings appear as shown in the following.

![Edit Asset: player.inputbindings](image)

h. In the **Edit Asset** window, choose **File, Save**.

**Step 6: Add Logic to the Player Script to Spawn the Projectile**

Now that you've set up the input binding, you can add logic to the **Player** script to spawn the projectile when you press the **Spacebar**.

**To add logic to the player script**

1. In Lumberyard Editor, choose **Tools, Script Canvas**.
2. In the **Script Canvas** editor, choose **File, Open**. Select `player.scriptcanvas` and then click **Open**.
3. Do the following to enable the event node to listen for the input event:
   a. In the **Node Palette**, enter **input** in the search box.
   b. Under **Gameplay, Input**, drag **Input Handler** from the **Node Palette** to the graph. **Input Handler** is an event node. When an event occurs, the event node sends a message to the script.
   c. For **Event Name**, enter **shoot**.

![Input Handler](image)

4. Do the following to control the firing rate for the shoot event:
   a. In the **Node Palette**, enter **once** in the search box.
   b. Under **Logic**, drag **Once** from the **Node Palette** to the graph.
c. From Input Handler, drag the Held pin to connect it to the In pin of the Once node.

d. In the Node Palette, enter `spawn` in the search box.

e. Under Gameplay, Spawner, drag Spawn from the Node Palette to the graph.

f. From Once, drag the Out pin to connect it to the In pin of the Spawn node.

g. In the Spawn node, for Source, enter the name of the child entity with the Spawner component attached from Step 4 (p. 2456).

h. In the Node Palette, enter `delay` in the search box.

i. Under Timing, drag Delay from the Node Palette to the graph.
j. From Spawn, drag the Out pin to connect it to the In pin of the Delay node.
k. In the Delay node, for Time, enter 0.25 (seconds).
l. From Delay, drag the Out pin to connect it to the Reset pin of the Once node.

**Note**
Node execution always flows from the left side to the right side of a node. For more information, see Inputs, Outputs, and Connection Types (p. 2497).

5. Verify that your player.scriptcanvas graph looks like the following.

**Example**

6. Choose File, Save. You can also press Ctrl+S.
7. In Lumberyard Editor, press Ctrl+G to enter game mode and test your script.
8. Do the following:
   - To move the sphere forward, press W.
   - To move the sphere backwards, press S.
   - To move the sphere from side to side, press A and D.
   - To spawn a projectile that propels forward, press the Spacebar.

     If you hold the Spacebar, the spawn fires once and the Delay node resets the Once node every 0.25 seconds.
9. When you are done testing your script, press Esc.

**Step 7: Create a Target Entity**

Now that the Player entity shoots projectiles, you can add a Target entity for the Player entity to destroy.
To create a target entity

1. In Lumberyard Editor, create an entity: Right-click in the Perspective viewport near your controllable sphere and choose Create entity.

2. In the Entity Inspector, for Name, enter Target.

3. Click Add Component and then choose the Mesh component.

4. In the Mesh component, for Mesh asset, click the browse (...) button, select the \SamplesProject\Objects\default\primitive_cylinder.cgf file, and then click OK.

5. Click Add Component and choose the following components:
   - Static Physics
   - Mesh Collider
   - Tag

6. In the Tag component, click + to add a tag and for [0], enter Target.
7. Verify that your **Target** entity appears like the following.

Example

![Image of Target entity settings](image-url)
Step 8: Add Logic to the Projectile to Destroy Target Entities

Now that you've set up the **Target** entity, you can add logic to the projectile to destroy entities that have the **Target** tag.

To add logic to destroy target entities

1. In the **Script Canvas** editor, open the `projectile.scriptcanvas` file.
2. Do the following to add collision nodes:
   a. In the **Node Palette**, enter `collision` in the search box.
   b. Under **Physics, Physics Component**, drag **On Collision** from the **Node Palette** to the graph. This node has a special data enter called `collision`.

3. Do the following to add a **Set hit** node:
   a. In the bottom-right pane, in the **Variable Manager**, click **Create Variable**. You can use variables to store and modify persistent values in your graph.
      For more information, see Managing Script Canvas Variables (p. 2482).
   b. Select **Collision**, double-click **Variable 1** and then rename the variable to **hit**.
   c. Right-click the **hit** variable and choose **Set hit**.
4. Select the **Out** pin for **On Collision** and drag to connect it to the **In** pin for the **Set hit** node.
5. From **On Collision**, drag the **Collision** pin to connect it to the **Collision** pin of the **Set hit** node.
6. Do the following to add tags:
   
a. In the **Node Palette**, enter **CRC32** in the search box.
   
b. Under **Math, CRC32**, drag **Create CRC32** from the **Node Palette** to the graph. The tag system uses CRC32s to store tag names.
   
c. From **Set hit**, drag the **Out** pin to connect it to the **In** pin of the **Create CRC32** node.
   
d. In the **Create CRC32** node, for **String**, enter **Target**.

![Create CRC32 Node]

   e. In the **Node Palette**, enter **tag** in the search box.
   
f. Under **Gameplay, Tag**, drag **Has Tag** from the **Node Palette** to the graph.

![Has Tag Node]

g. From **Create CRC32**, drag the **CRC32** pin to connect it to the **Tag** pin of the **Has Tag** node.

7. Do the following to add a **Get hit** node:

   a. In the **Variable Manager**, right-click the **hit** variable and choose **Get hit**.
   
   b. From **Get hit**, drag the **entity: EntityID** pin to connect it to the **Source** pin of the **Has Tag** node. This enables the **Has Tag** node to check the entity that the projectile collided with rather than itself.
   
   c. From **Create CRC32**, drag the **Out** pin to connect it to the **In** pin of the **Get hit** node.
   
   d. From **Get hit**, drag the **Out** pin to the **In** pin of the **Has Tag** node.
8. Do the following to add the **If** node:

   a. In the **Node Palette**, enter `if` in the search box.
   b. Under **Logic**, drag **If** from the **Node Palette** to the graph.
   c. From **Has Tag**, drag the **Out** pin to connect it to the **In** pin of the **If** node.
   d. From **Has Tag**, drag the **Result** pin to connect it to the **Condition** pin for the **If** node.

9. Do the following to add a **destroy** node:

   a. In the **Node Palette**, enter `destroy` in the search box.
   b. Under **Entity, Game Entity**, drag **Destroy Game Entity and Descendants** from the **Node Palette** to the graph.
c. From If, drag the True pin to connect it to the In pin of the Destroy Game Entity and Descendants node.

d. From Get hit, drag the entity: EntityID pin to connect it to the EntityID pin of the Destroy Game Entity and Descendants node.

e. Verify that your projectile.scriptcanvas graph looks like the following.

Example

![Diagram of projectile.scriptcanvas graph]

10. In the Script Canvas editor, choose File, Save. You can also press Ctrl+S.

11. In Lumberyard Editor, press Ctrl+G to enter game mode and test your script.

12. To aim and shoot at the target, do the following:

   • To move the sphere forward, press W.
   • To move the sphere backward, press S.
   • To move the sphere from side to side, press A and D.
   • To spawn a projectile that propels forward, press the Spacebar.

13. When you are done testing your script, press Esc.

More Learning Opportunities with Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Here are some additional educational opportunities for learning more about Script Canvas.

Sample Levels

To help you get started with Script Canvas, the following sample levels are available:

• For a basic sample level, see Script Canvas Basic Sample (p. 149), which is included in your Lumberyard setup. The sample uses a tank to fire a projectile at a target.
• For a more in-depth sample level, install the Project N.E.M.O. (p. 2467) sample level, which uses a third-person camera to control a mini sub through an underwater mine field.

Video Tutorials

These video tutorials in the Tutorial Guide are also available to assist in your journey towards becoming a Script Canvas expert:

• Tour of Script Canvas (2 videos)
• Explosive Red Barrel (4-part series)
• Tank Demo (3-part series)
• Using Trigger Areas and Capturing Player Input (5-part series)

Installing the Project N.E.M.O. Sample

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Project N.E.M.O (Nautical Emergency Maneuvers and Operations) is a complete sample game level that showcases some of the key capabilities of Script Canvas. The sample level creates a short underwater navigation experience in which the player maneuvers a mini sub through obstacles and collects pickups. The sample level uses Script Canvas and script events to implement submarine functionality, mines and squid behavior, HUD display, pickups, world space UI, and game end.

To use the sample, you must first install and configure Lumberyard. Then download, configure, and build the project.

Step 1: Install and Configure Lumberyard

The following procedure shows you how to install Lumberyard and configure it for the Project N.E.M.O. sample level. If Lumberyard is already installed, start with step 2 to configure it.

To install and configure Lumberyard for the Project N.E.M.O. sample level

1. Download and run the Lumberyard installer.
2. Do one of the following:
   • If you are installing Lumberyard, click Launch Lumberyard Setup Assistant.
   • If Lumberyard is already installed, click the Lumberyard Setup Assistant icon on your desktop.
3. Click Customize.
4. In Lumberyard Setup Assistant, ensure that Create, Modify and Build projects is selected.
5. For Visual Studio Version, select the version of Visual Studio that you want to use.
6. Click Install required SDKs.
7. For Required SDKs, click Install all.
8. After the required SDKs are installed, close Lumberyard Setup Assistant.

Step 2: Download, Configure, and Build the Project

Now you are ready to download, configure, and build the sample project.

To download, configure, and build the Project N.E.M.O. sample level

1. Download the NEMO.zip file.
2. Extract the NEMO and Engine folders from the .zip file to the \texttt{lumberyard\_version\dev\} directory.
3. On your desktop, click the Project Configurator icon.
4. In Project Configurator, click the NEMO project to select it, and then click Set as default.
6. From a command window on \texttt{lumberyard\_version\dev\} directory, run the following command:
   
   
   \texttt{lmbr\_waf configure}

7. Run the version of the following command that corresponds to your version of Visual Studio.

   \texttt{lmbr\_waf build\_win\_x64\_vs2017\_profile -p game\_and\_engine}

   \textbf{Note}
   Building the project takes some time.

At this point, the Project N.E.M.O. sample level is ready. For more information about the sample and how it uses Script Canvas, see the Amazon Game Tech Blog.

**Common Tasks**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The topics in this section provide practical knowledge about common tasks associated with Script Canvas, including best practices, advice on troubleshooting, and use of the debugger.

**Topics**

- Adding Scripts to Entities (p. 2468)
- Script Canvas Best Practices (p. 2469)
- Script Canvas Errors and Troubleshooting (p. 2470)
- Script Canvas Debugging (p. 2471)

**Adding Scripts to Entities**

Like Lua scripts, you can add a script to a level or slice is by adding a Script Canvas component to an entity. The Script Canvas component adds Script Canvas capabilities to the entity and provides a field to specify a script assignment. You can specify a script with the Script Canvas component.

For more information, see the Script Canvas (p. 818) component.

For more information about adding a component to an entity, see Adding Components to an Entity (p. 479).
Script Canvas Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Best practices for Script Canvas include using an event-driven approach and using custom nodes to simplify your graphs.

Use an Event-Driven Approach

Script Canvas nodes are by default stateless. However, by connecting to the TickBus (p. 1019), they can be configured to have a state. Engineers must manage the lifetime and performance of nodes that have a state.

In the core Script Canvas libraries, state is used primarily to drive the execution of the graph (as with the Delay node). However, an event-driven paradigm is recommended because it helps reduce the complexity of authoring and executing graphs.

We recommend that you reflect functionality to Script Canvas through the behavior context. This is true even for Script Canvas-specific functionality. Using the behavior context encourages event-driven paradigms through EBuses. This approach yields modular, decoupled behaviors that can reduce graph complexity and takes advantage of execution optimizations.

Use Custom Nodes to Simplify Your Graphs

Identify frequently used but complicated user patterns and simplify them through custom nodes and/or improved behavior context methods. Using custom nodes with EBuses can reduce the overall complexity of graphs and make graph authoring more intuitive. For information on creating custom nodes, see Creating Custom Nodes in Script Canvas (p. 2553).

Be Careful with Entity Activation Order

Sending events during entity activation can have undesired results. Because the order of activation of entities is not guaranteed, when an event is sent during activation, some entities that need to handle the event might not receive it. In particular, the On Graph Start and On Entity Activated nodes are subject to activation order issues. Be careful when sending events from them.

In order to ensure that all entities that need to listen for and handle a given script event are ready to receive the event, it is best to queue the message on the tick bus. To implement this strategy, use a Once node connected to the On Tick message, as the following image shows. This practice guarantees that when the message is sent, all entities that might be connected to that script event receive it.
Script Canvas Errors and Troubleshooting

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas provides a mechanism to detect and report errors. Errors in Script Canvas are generally caused by invalid conditions.

In C++ Script Canvas provides the following macros for error management:

```c
#define SCRIPTCANVAS_NODE_DETECT_INFINITE_LOOP(node)  
#define SCRIPTCANVAS_RETURN_IF_NOT_GRAPH_RECOVERABLE(graph)  
#define SCRIPTCANVAS_HANDLE_ERROR(node)  
#define SCRIPTCANVAS_REPORT_ERROR(node, ...)  
#define SCRIPTCANVAS_RETURN_IF_ERROR_STATE(node)  
```

You can use these macros during a graph's execution. When triggered, they stop the execution of nodes that have an error.

You can detect errors that occur in a graph at run time and provide an error handler to respond to the errors. This helps users to gracefully handle graphs that encounter an invalid condition.

Runtime execution errors are reported to the Event Handler node. You can use this node to perform custom logging or take appropriate action when an error occurs.
You can also raise and handle errors within a graph's execution. The following simple example shows how you could use Error and Error Handler nodes to provide notification of an error in data validation. This helps to rectify the solution to ensure that the graph remains stable.

Script Canvas Debugging

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas supports live debugging of a Script Canvas graph running in-game. You can use Lumberyard Editor or non-editor tools like game launchers as debug targets.

To open the Script Canvas debugger and choose a target

1. From the Script Canvas editor, choose Tools, Debugging. The debugger panel opens at the bottom of the Script Canvas editor.

2. On the Live tab of the debugging panel, use the dropdown menu to choose the debug target. The default target is Lumberyard Editor, but you can use Script Canvas to debug the game running in standalone mode. For more information, see In-Game Debugging (p. 2478).
Choosing Entities and Graphs to Debug

After you choose a debug target, you can choose the entities and graphs that you want to debug.

To choose entities and graphs to debug

1. To see entities with Script Canvas graphs that are available for debugging, expand the items on the Entities tab. The Entities tab displays the entities known to the debugger at edit time that have Script Canvas graphs.

   ![Entities Tab Screenshot]

   **Note**
   Running the same graph multiple times on a single entity is not supported.
2. To get a complete listing of all available Script Canvas graphs in the project, click the **Graphs** tab. Each graph in the table shows all entities that are using that graph. The **Graphs** tab is useful for debugging dynamically spawned scripts. For more information, see **Debugging a Dynamically Spawned Graph (p. 2478)** later in this topic.

![Graphs tab screenshot](image)

3. On the **Entities** tab or **Graphs** tab, select the check boxes for the entities or graphs that you want to debug.
4. To capture all instances of a selected graph, select **All Graph Instances**.

**Configuring Debugger Options**

Use the following options to configure debugger behavior.
Auto Capture

Enable this option if you want the output from a specified target to be captured as soon as the debugger connects to it. For external tools, capture starts immediately when you enable this option. For Lumberyard Editor, capture starts when the editor enters game mode.

Live Updates

Enable this option to display the data as it is captured. When disabled, the data is captured silently and displays only after the capture completes.

Note
When live updates are enabled and you capture a large amount of data, editor performance decreases noticeably. For better performance, you should disable live updates, especially for longer captures.

Running the Debugger

After you have chosen the entities or graphs to debug, you are ready to run the Script Canvas debugger.

To run the Script Canvas debugger

1. Click Capture. The Capture button puts Lumberyard into gameplay mode automatically.

Note
If Editor is selected as the capture target, the game must be running for the debugger to return results.

The Script Canvas debugger begins capturing data when the graph runs. If live updates are enabled, the data appears in the debugger panel when the graph being debugged becomes active during gameplay. Otherwise, the data appears after the capture completes.
2. After you have enough data, click **Capture** again to stop the data capture.

**Examining Captured Data**

The captured data is presented in a log that is sorted in order of processing. Each line represents the processing of a single node.

![Log of captured data](image)

**Note**

Currently, only a single instance of captured data can be stored. Previous data is lost when a new set of data is captured.

**To examine the captured data**

1. To see the Script Canvas node that corresponds to a line in the log, click the line.

   Each line typically shows the node's **In** signal and **Out** signal. If the **In** or **Out** signal is not present, the node is either the first or final node of a given execution line. As shown in the following image, the **Set Location Rotation** node is the final node in the execution line, so an **Out** signal is not present.
2. Use the up or down arrow keys to move through the log messages in the debugger panel. As you do so, the corresponding node to in Script Canvas graph is highlighted.

3. To examine the data that the node was using, expand the log message.

   **Note**
   Some nodes send additional information that appears in the form of annotations. For example, the **Print** node sends the full string that it displayed.

4. To expand all lines, click the **expand icon**.

5. To collapse all lines, click the **collapse icon**.

6. To search for a particular node name or names, use the **Search box**.
Common Tasks

Debugging a Dynamically Spawned Graph

Dynamically spawned graphs are usually part of a dynamic slice. Because dynamically spawned graphs cannot be known at edit time, you must select them by name rather than by the entity or entities on which they are used.

To debug a dynamically spawned graph

1. On the Graphs tab, select the name of the graph.
2. Follow the same steps that you use to debug any other Script Canvas graph. The debugger records the graph’s operations when the graph becomes active during gameplay.

In-Game Debugging

For in-game debugging, you use the Script Canvas debugger to connect to a running game launcher.

To debug a running game

1. Run GridHub.exe from the Bin64vcxxx directory that corresponds to the version of Visual Studio that you are using (for example, lumberyard_version\dev\Bin64vc141\GridHub.exe). GridHub is the network environment that provides connectivity between Lumberyard and its tools. For more information, see Using GridHub (p. 1954).

   Note
   GridHub must be active for non-editor targets to appear in the Live tab.

2. Run the launcher for your game (for example, lumberyard_version\dev\Bin64vc141\SamplesProjectLauncher.exe).
3. On the Live tab of the Script Canvas debugger, choose the launcher from the list of debug targets. When you choose the launcher as the debug target, Script Canvas execution is recorded for the graphs that you specify.
Notes

When using the Script Canvas debugger, keep the following points in mind.

Performance

The editor experiences a severe decrease in performance when it is capturing data. To mitigate this, disable live updates and avoid capturing lengthy sessions. Lengthy sessions can easily lead to a sharp increase in logging events. If your game runs at 60fps and you have 40 Script Canvas nodes that run on each tick, 2400 log messages must be displayed every second. After a minute, this number increases to 144,000 messages. To minimize the amount of data captured, limit the scope and intensity of logging.

Saving Issues

When you modify a graph and save it, some IDs are remapped in the asset, but not in the visualized Script Canvas scene. As a result, the unified ID used in the logging messages no longer matches the visual presentation. This mismatch causes the visual scraping to fail. To work around this, close and re-open the Script Canvas scene.

Editor Reference

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Script Canvas editor provides a visual scripting environment for creating graphs and reusable script functions. In the following topics, learn more about each menu option and editor tool, as well as valuable keyboard and mouse shortcuts to increase your productivity.

Topics

- Script Canvas Editor Shortcuts (p. 2480)
- Managing Script Canvas Variables (p. 2482)
- Working with Nodes (p. 2496)
- Script Canvas Functions (p. 2538)
# Script Canvas Editor Shortcuts

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following table shows the keyboard shortcuts that you can use in the Script Canvas editor.

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrow Keys</td>
<td>Scroll graph</td>
<td>Scrolls the graph left, right, up, or down.</td>
</tr>
<tr>
<td>Ctrl+C</td>
<td>Copy</td>
<td>Copies the selected nodes and their connections to the clipboard.</td>
</tr>
<tr>
<td>Ctrl+V</td>
<td>Paste</td>
<td>Pastes copied nodes and their connections from the clipboard into the active graph.</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Duplicate</td>
<td>Duplicates the selected nodes in the active graph. This is the equivalent of using Ctrl+C and Ctrl+V.</td>
</tr>
<tr>
<td>Ctrl+Left Arrow</td>
<td>Select inputs</td>
<td>Selects all nodes that are connected to the input pins of the currently selected node.</td>
</tr>
<tr>
<td>Ctrl+Right Arrow</td>
<td>Select outputs</td>
<td>Selects all nodes that are connected to the output pins of the currently selected node.</td>
</tr>
<tr>
<td>Ctrl+Up Arrow</td>
<td>Select connected nodes</td>
<td>Selects all nodes that are connected to the currently selected node.</td>
</tr>
<tr>
<td>ESC</td>
<td>Clear selection</td>
<td>Deselects any selected nodes.</td>
</tr>
<tr>
<td>Ctrl+Shift+P</td>
<td>Screenshot</td>
<td>Creates an image of the area around all selected nodes and adds it to the clipboard. If no nodes are selected, an image of the entire active graph is added to the clipboard.</td>
</tr>
<tr>
<td>Shift+Left Arrow</td>
<td>Align left</td>
<td>Aligns all the selected nodes along a left edge.</td>
</tr>
<tr>
<td>Shift+Right Arrow</td>
<td>Align right</td>
<td>Aligns all the selected nodes along a right edge.</td>
</tr>
<tr>
<td>Shift+Up Arrow</td>
<td>Align top</td>
<td>Aligns all the selected nodes along a top edge.</td>
</tr>
<tr>
<td>Shift+Down Arrow</td>
<td>Align bottom</td>
<td>Aligns all the selected nodes along a bottom edge.</td>
</tr>
<tr>
<td>Ctrl+Alt+M</td>
<td>Add comment</td>
<td>Adds a new comment using the properties from the default comment preset. For information about presets, see Creating Comment and Group Presets (p. 2530). <strong>Note:</strong> NVIDIA's GeForce Experience overlay uses a default setting for turning on/off the microphone that interferes with this hotkey.</td>
</tr>
<tr>
<td>Ctrl+Shift+G</td>
<td>Group selection</td>
<td>Groups the selected nodes on the graph using the properties from the default node group preset. For information about presets, see Creating Comment and Group Presets (p. 2530).</td>
</tr>
<tr>
<td>Ctrl+Shift+H</td>
<td>Ungroup</td>
<td>Ungroups the currently selected group.</td>
</tr>
</tbody>
</table>
### Key Combination

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + Number_Key</td>
<td>Create bookmark</td>
<td>Creates a bookmark out of the current view and assigns it to the specified number key. If you choose a number that is already assigned to a bookmark or a bookmark-enabled group, you are prompted to reassign the existing bookmark. For more information about bookmarks, see Adding Bookmarks for Script Canvas (p. 2534). For information about enabling groups as bookmarks, see Grouping Nodes (p. 2526).</td>
</tr>
<tr>
<td>Number_Key</td>
<td>Jump to bookmark</td>
<td>Jumps to the bookmark location associated with the key that is pressed.</td>
</tr>
<tr>
<td>Ctrl + Plus Sign (+)</td>
<td>Zoom in</td>
<td>Zooms the graph in.</td>
</tr>
<tr>
<td>Ctrl + Minus Sign (-)</td>
<td>Zoom out</td>
<td>Zooms the graph out.</td>
</tr>
<tr>
<td>Ctrl + Shift + Up Arrow</td>
<td>Zoom to selection</td>
<td>Centers the view on the nodes that are currently selected.</td>
</tr>
<tr>
<td>Ctrl + Shift + Down Arrow</td>
<td>Show entire graph</td>
<td>Centers the entire graph into the current display. Zooms out as much as possible to display all nodes.</td>
</tr>
<tr>
<td>Ctrl + Shift + Left Arrow</td>
<td>Show start of chain</td>
<td>Centers the view on the nodes that do not have any input connections, and are connected to the selected node through their output connections.</td>
</tr>
<tr>
<td>Ctrl + Shift + Right Arrow</td>
<td>Show end of chain</td>
<td>Centers the view on the nodes that do not have any output connections, and are connected to the selected node through their input connections.</td>
</tr>
<tr>
<td>Ctrl + K, Ctrl + C</td>
<td>Comment out selected nodes</td>
<td>Comments out the current selection of nodes and turns them gray. Commented out nodes are not run at runtime, but still exist at edit time.</td>
</tr>
<tr>
<td>Ctrl + K, Ctrl + U</td>
<td>Uncomment selected nodes</td>
<td>Uncomments the selected nodes.</td>
</tr>
</tbody>
</table>

**Note**

If a keyboard shortcut doesn’t appear to work for you, another process running in the background might have bound that key combination. Consider searching the forums to see if others have encountered this issue.

### Mouse Shortcuts

The following shortcuts use the mouse or keyboard and mouse.

<table>
<thead>
<tr>
<th>User Action</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt + Left Click</td>
<td>Disconnect and delete a single node or group</td>
<td>Disconnects and deletes the node or group clicked.</td>
</tr>
<tr>
<td>Alt + Left Click</td>
<td>connection</td>
<td>Deletes the connection clicked from the active graph.</td>
</tr>
<tr>
<td>Alt + Left Click</td>
<td>slot</td>
<td>Deletes any connections to the slot from the active graph.</td>
</tr>
<tr>
<td>User Action</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Ensure that the connections that you want to delete are highlighted before pressing <strong>Alt+Left Click</strong>. Otherwise, you might delete the node instead.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Middle Mouse Button (Scroll Wheel) Click graph tab</td>
<td>Close graph</td>
<td>Closes the open graph that corresponds to the tab that you clicked. If the graph has changed and has not been saved, you're prompted to save it first.</td>
</tr>
<tr>
<td>Scroll mouse wheel</td>
<td>Zoom the graph</td>
<td>Zooms the graph in or out.</td>
</tr>
</tbody>
</table>

### Managing Script Canvas Variables

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Variable Manager** shows the variables that are used in your Script Canvas graph. These variables represent the custom data required to build game logic. For example, you can use variables to make counters, store entity references, specify a direction, or define a color.

**Topics**
- Adding and Configuring Variables (p. 2482)
- Setting Default Values for Variables (p. 2484)
- Creating Get or Set Variable Nodes (p. 2485)
- Creating Value-Changed Nodes (p. 2486)
- Deleting Variables (p. 2487)
- Using Container Types (Arrays and Maps) (p. 2489)

### Adding and Configuring Variables

You can add variables to your Script Canvas graph to declare and initialize them.

**To add and configure a variable**

1. In the **Script Canvas** editor, open your Script Canvas graph or create one.
2. In the **Variable Manager**, click **Create Variable** and then choose your variable type. You can search to filter the list of variable types.

**Note**
Common variable types are pinned to the top of the list by default. You can customize the pinned list to show the variable types that you use most often. To do so, click the box to the left of a variable type to pin or unpin it.
3. In the **Node Inspector**, configure the properties for your variable.
For example, if you add a **Color** variable, you can do the following:

- **For Name**, enter a name to identify that color variable. You can also double-click the name in the **Variable Manager** to rename the variable.
- **For Color**, enter an RGB value or use the color picker.
- **For Display Order**, enter the relative order in which you want the variable to appear in the Script Canvas editor, or leave the default at -1.
- **For Scope**, select **In** to show the variable property and value under the assigned [Script Canvas (p. 818)](#) component in the **Entity Inspector**, or leave the default at **Local** to keep the variable private to the graph.

**Note**

This setting allows you to use the same Script Canvas graph for more than one entity, but customize part of the graph for a specific entity. When you change the variable value on a component, that value takes precedence over the default value that is specified in the graph.

4. In the **Script Canvas** editor, choose **File, Save** to save your changes.

## Setting Default Values for Variables

You can set the default value for variables in the **Node Inspector** or the **Variable Manager**.

### To set variable values

1. In the **Variable Manager**, select the variable that you want to update. You can search to filter the list of variables that are in your graph.
2. Do one of the following to update the properties for your variable:

   - **In the Node Inspector**, update the variable values as needed.

   - In the **Variable Manager**, update certain variable values. These values appear in a third column and can be selected or double-clicked.
3. In the **Script Canvas** editor, choose **File, Save** to save your changes to the graph.

**Creating Get or Set Variable Nodes**

You can use get and set variable nodes to retrieve or set the variable's value.

**To create get or set variable nodes**

- Do one of the following:
  - Drag the variable from the **Variable Manager** to the canvas, and then choose **Get variable name** or **Set variable name**.
  - Right-click the variable in the **Variable Manager** and choose **Get variable name** or **Set variable name**.
• Use the following keyboard shortcuts:
  • Press **Shift** and drag the variable from the **Variable Manager** to the canvas to create a get variable node.
  • Press **Alt** and drag the variable from the **Variable Manager** to the canvas to create a set variable node.

### Creating Value-Changed Nodes

You can use **OnVariableValueChanged** (value-changed) event nodes to react to a change in a variable's value.

**To create value-changed nodes**

• Do one of the following:
  • Drag the variable from the **Variable Manager** to the canvas, and then choose **On** variable name **Changed**.
  • Create a new **OnVariableValueChanged** event node in your graph and set the **Source** field to a variable using the field's gear button. For help adding a node to a graph, see **Adding and Connecting Nodes** (p. 2506).
Deleting Variables

You can delete variables from the graph or the Variable Manager.

To delete a variable

- Do one of the following:
  - Select the variable node in the canvas and press Delete.
  - Select the variable in the Variable Manager and press Delete.
  - Right-click the variable in the Variable Manager and choose Delete variable name.
Variable Manager

Create Variable

Search...

- Blue Color
- Get Direction
- Green Color
- Move Entity
- Purple Color

Get Green Color
Set Green Color
Delete Green Color
Using Container Types (Arrays and Maps)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas has support for arrays and maps.

Arrays

Arrays provide a dynamic continuous area of memory that can hold storage of a given type.

To create an array variable

1. In the Variable Manager, do one of the following:
   - Click Create Variable, and then click Array.
   - In the Variable Type search box, type Array, and then click Array.

2. Enter information to create your array, and then click Create.

   - For Variable Name, enter the name for your array variable.
   - For Container Type, use Array.
   - For Type, choose the data type for the array.
   - (Optional) To pin the array to the list of variables in the Variable Manager, select Pin To Variable List. Then, when you click Create Variable, the array appears in the list as Array<\text{data\_type}>. This is useful when you reuse the same type frequently.
Array Pin Icons

Some nodes, like **OnEntitiesSpawned** or **Get String Array**, provide data as arrays. The data pins for the arrays on such nodes have a square icon. The color of the icon shows the data type for the array.
Array Operation Nodes

In the Node Palette, the Containers section has nodes that you can use to add, get, and remove elements from arrays.

- **Add Element at End** – Adds the element at the end of the container.
- **Clear All Elements** – Removes all elements from the container.
- **Erase** – Erases the element at the specified index or key.
- **For Each** – Iterates through each element of a container.
- **Get Element** – Returns the element at the specified index or key.
- **Get First Element** – Returns the first element in the container.
- **Get Last Element** – Returns the last element in the container.
- **Get Size** – Returns the number of elements in the container.
- **Insert** – Inserts an element into the container at the specified index or key.
- **Is Empty** – Returns whether the container is empty.

All container operation nodes have an In pin and a Source pin.

All pins have an Out pin except For Each, which has an Each pin that is signalled after each element in the container. The For Each node also has a Break pin and a Finished pin. The Break pin stops the iteration when signalled. The Finished pin is signalled when all of the elements in the container have been iterated over or the iteration is stopped by the Break pin.

Automatic Data Pin Typing

The pins on container operation are context-sensitive. Data input and output pins automatically take on the data type of the pin to which they are connected.
Example

In the following example, the array type is `string`.

When you connect the `Array<String>` pin to the Source pin on the Add Element at End node, the following changes occur automatically:

- The Source pin changes to the string type.
- The Add Element at End node box expands to include a String text box where you can enter a string value.
- The pin icon and line colors change to the color of the data type that you are using (in the case of this example, blue).
- A Container pin for chaining output appears on the target node.

Chaining Array Operation Nodes

Array operation nodes typically have a Container output pin. You can use this output pin to chain several operations on the same array, as the following image shows.
Maps

Maps are containers of key-value pairs. Maps in the **Variable Manager** have a small split rectangular icon.

To create a map variable

1. In the **Variable Manager**, do one of the following:
   - Click **Create Variable**, and then click **Map**.
   
   or
   
   - In the **Variable Type** search box, type **Map**, and then click **Map**.

2. Enter information to create your map, and then click **Create**.
For **Variable Name**, enter the name for your map variable.

For **Container Type**, use **Map**.

For **Key**, choose a data type for the key.

For **Value**, choose a data type for the value.

(Optional) To pin the map to the list of variables in the **Variable Manager**, select **Pin To Variable List**. Then, when you click **Create Variable**, the map appears in the list as **Map<key_data_type,value_data_type>**. This is useful when you reuse the same key pair combination frequently.

**Map Pin Icons**

You can use **Get** or **Set** nodes to get values from a map or make changes to the map. The data pins for the maps have square icons with two colors. One color represents the data type for the key, the other for the value.
Map Operation Nodes, Data Pin Typing, and Chaining

Maps use the same container operation nodes as arrays. As with arrays, data pins automatically take on the data types of the map to which they are connected, as the following image shows.

As with arrays, maps provide a data output pin that you can use to chain operations on the same map.
Working with Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Games are often built with complex logic, and complex logic can produce a large number of nodes and connections. Script Canvas provides some specialized tools to help you manage this complexity.

Topics
- Understanding Script Canvas Nodes (p. 2496)
- Adding and Connecting Nodes (p. 2506)
- Adding Variable References in Script Canvas Nodes (p. 2511)
- Setting Entity Targets (p. 2514)
- Deleting Nodes (p. 2515)
- Disconnecting Nodes (p. 2516)
- Organizing Nodes (p. 2517)

Understanding Script Canvas Nodes

A node in Script Canvas consists of a title bar, inputs, and outputs.
**Title Bar** – The title bar of a node is in a colored band at the top of the node. A title bar can include a subtitle, but not all nodes have subtitles.

**Inputs** – Located on the left side of the node. An input can be an execution, or logic, input or a data input. Execution inputs cause the node to act. Data inputs provide the node with the data it requires to do processing or decision making.

**Outputs** – Located on the right side of the node. Outputs are also either execution or data related.

**Inputs, Outputs, and Connection Types**

Lumberyard has two main pin and connection types. Some inputs and outputs determine the flow of logic and order of execution. Other inputs and outputs pass data from one node to the next.

**Logic inputs, outputs, and connections**

The execution of a script is driven by triangular inputs and outputs on every node. These connections determine the order of execution. A *Script Canvas* script runs when the entity that is attached to the script is activated. Nodes are connected from the inputs on their left side. After they finish running, they activate nodes that are connected to their outputs on the right side.

An output logic pin with multiple connections runs a logic branch in sequence. The execution sequence is determined by the order that the connections were made, from earliest to most recent. If a specific execution order is required, you can specify the sequence order by using a single logic flow or a *Sequencer* node.

An incoming logic pin with multiple connections runs each time that the logic flow triggers the node. For example, if a node is triggered by three different nodes in a script, the node runs three times.

**Data inputs, outputs, and connections**

Data connections enable scripts to read and write data between nodes. Data is read from the right side of one node and then set on the left side of another node.

**Making Connections**

Connections can be made only between pins of the same type. For example, logic connections can be made only between logic pins, and data connections can be made only between data pins of the same type. You can't create connections between incompatible pins, such as logic and data.

**To make a connection**

1. In the *Script Canvas* editor canvas, drag from the input pin of one node to an output pin of another node. This creates a connection line between the two pins.

2. To move a connection from one pin to another, drag the end of a line from one pin and drop it onto another pin.

   To delete a connection, right-click and choose *Delete*. You can also press and hold *Alt*, and click the connection to delete it.

**Variable Nodes**

Variable nodes enable *Script Canvas* to read from or write to specific variables.
Another way to read or write a variable is to use variable references on a node data pin.

For more information about using variables in Script Canvas, see Managing Script Canvas Variables (p. 2482).

For more information about creating variable references, see Adding Variable References in Script Canvas Nodes (p. 2511).

**Event Nodes**

In Lumberyard's Event Bus (EBus) system (p. 1851), events can be sent or received, so Script Canvas has sender nodes and receiver nodes.

**Sender Nodes**

Event senders send events directly to a specific entity or broadcast events to all entities that are listening for and interested in handling the event. Most events are addressable, which means they can be sent to a
specific entity. Because events are usually sent to entities, the most common address type is Entity Id, although other address types can be used.

The following example uses Light events to create a sender node.

**To create a sender node**

1. In the **Node Palette** search box, type **Light**. The results show nodes related to light.

   ![Node Palette screenshot]

   In the **Node Palette**, sender events are the dark blue entries. All Light-related sender events provide a way to communicate with, configure, or alter the behavior of a given Light component. You can send any of the Light-related sender events to an entity that has a Light component. If the entity that owns the Script Canvas graph also has a Light component, it can send the event to itself.

2. Drag **Turn On** or **Turn Off** onto the canvas to create a sender node.
The **Source** pin of the sender node refers to the entity that sends the event. The default is **Self**, which means that it sends Light events for the same entity that the Script Canvas component is on. However, you can change the source to any entity in the game world.

The **State** pin is a Boolean value that controls the state of the light.

**Receiver Nodes**

An event receiver implements a particular behavior when it receives a particular event.

The following example creates a receiver node for a Light event.

**To create a receiver node**

1. In the **Node Palette** search box, type **Turn**.
In the list of results, event receivers like **Turned Off** and **Turned On** have a light blue icon.

2. Drag **Turned On** onto the canvas to create a receiver node.

The **Source** pin of the receiver node refers to the entity from which the event is received. The default is **Self**, which means the node receives Light events for the same entity that the Script Canvas component is on. You can change the target to any entity in the game world.

In Lumberyard v1.24 and later, you can also specify the target using a variable reference (p. 2511). Whenever the variable changes, the EBus handler will update the Source to match the variable reference.

3. Click **Add/Remove Events**.

Because receiver nodes are usually containers for multiple events, you can click **Add/Remove Events** to view and add any of the available event receivers for a given component. In this case, the Light component exposes two events: **Turned Off** and **Turned On**.
4. Select the **Turned Off** check box to add the **Turned Off** event to the receiver node.

A second blue band in the node appears. The node is now listening for both the **Turned On** and **Turned Off** events.
5. Click **Add/Remove Events** again, and clear the **Turned Off** check box. The **Turned Off** event is removed from the receiver node.

### Displaying and Using Connection Controls

All receiver nodes have connection-related pins, or controls, that are hidden by default. You can use these controls to manage when an event is connected or disconnected (connected means that the event is ready to receive events, and disconnected means that the event is not receiving events). The connection controls can also notify you when a node successfully connects, disconnects, or experiences an error.

The following example uses the Light component **Turned On** event node.

**To enable and use Display Connection Controls**

1. Ensure that **Node Inspector** is visible. In Script Canvas editor, choose **View, Node Inspector**, or press **Ctrl+Shift+I**.
2. Click the Light **Turned On** node to select it.
3. In **Node Inspector**, select **Display Connection Controls**.

The Light component **Turned On** receiver node expands to provide connection-related pins.

- **Connect** and **Disconnect** – Use the **Disconnect** pin to prevent the receiver node from connecting. When the event should be connected and available to receive events, use the **Connect** pin.
The **Connect** and **Disconnect** pins are especially useful when working with the **On Tick** event. For example, if you have a complex operation that you do not want processed for every tick of the game, you can disconnect the **On Tick event** until it is required.

**Note**
When you enable a receiver node's **Display Connection Controls** property, the node no longer connects automatically. In this case, Script Canvas assumes that you want to specify when the connection occurs.

- **OnConnected** – Triggered when the event connects successfully. This pin is useful if you want to continue execution along the connection path when the connection occurs.
- **OnDisconnected** – Triggered when the event disconnects successfully. This pin is useful if you want to continue execution along the disconnection path when the connection occurs.
- **OnFailure** – An event fails to connect if it requires a source and no source is provided, or an invalid source is provided. The **OnFailure** pin displays diagnostic information that you can use to verify whether address data was correctly specified to the **Source** pin of the receiver node.

**Example**

In the following example, **Display Connection Controls** is enabled for the **On Tick** event receiver node. The **Tick** event is disconnected at the start of the graph's lifetime. When the light is turned on, the example changes the light's color randomly for every tick.
Adding and Connecting Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add nodes to a canvas by dragging and dropping them from the Node Palette onto the canvas. You can also quickly add and connect nodes without using the Node Palette, as described in the following sections.

Creating a Chain of Linked Nodes

You can add nodes quickly in succession to a graph by using the daisy chaining feature. Script Canvas automatically links the pins on each new node to the previous node for you.

To create a chain of linked nodes

1. On the right side of the graph, press Shift + Right-click.
2. In the context menu search box, enter the name or partial name of the node that you want to add.
3. Press Enter to accept the search result, or click the name of another node in the result list.

The new node appears on the graph with a line extended from the output pin. The context menu opens automatically, ready for you to add another node and link it to the previous one.
4. Continue to add as many connected nodes as you want. Previously created nodes automatically scroll off to the left.

**Note**
If you start by adding nodes on the left side of the graph, the nodes that you create quickly scroll out of view. To see the previous nodes that you created, start by pressing Shift + Right-click on the right side of the graph.

5. To exit the daisy-chaining mode, press ESC.

**Creating a Node From the Output Pin of an Existing Node**

**To create a connected node from an output pin of an existing node**

1. From an output pin on the existing node, drag a line onto the canvas.
2. In the context menu search box, enter the name or partial name of the node that you want to add.
3. Press Enter to accept the search result, or click the name of another node in the result list. The pins on the new node connect automatically to the pins on the existing node.
4. To create a chain of linked nodes from the output pin of an existing node, press Shift and drag a line from the output pin onto the canvas.

**Inserting a Node Between Two Connected Nodes**

To insert a node between two connected nodes and connect the new node automatically, you can use the following methods:

- Insert an existing node between the nodes.
- Create a new node between the nodes.

**To insert an existing node between two connected nodes**

1. Drag a node over the line that connects the two nodes, and hold the new node in position.
2. When an expanding, box-shaped animation appears, the node pins are connected. Script Canvas nudges surrounding nodes aside to accommodate the new node.
3. The node-nudging feature is enabled by default. To change it, choose **Edit, Settings, Global Preferences** and select or clear the **Allow Node Nudging On Splice** option.

![Global Preferences](image)

To create a node between two connected nodes

1. Right-click the line that connects the two nodes.
2. In the context menu search box, do one of the following:
• Enter the name of the node that you want, and press Enter.
• Choose a node from the list.

The pins on the new node connect automatically to the nodes on the left and the right.

Connecting Two Existing Nodes Automatically

You can connect two nodes by dragging one node onto another node. Script Canvas connects the corresponding pins for you.

To connect two nodes automatically

1. Drag a node over the side of the node that you want to connect, briefly holding the new node in position. For example, drag the left side of one node onto the right side of the second node.
2. When an expanding, box-shaped animation appears over the overlapping nodes, the node pins have been connected.
3. Adjust the position of the new node on the graph.
Adding Variable References in Script Canvas Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Introduced in v1.24 of Lumberyard, variable references provide a shortcut for getting and setting variable values directly from the Script Canvas nodes that use them. Any data pin can be converted to a variable reference by dragging a variable from the Variable Manager directly onto the pin.

To create a variable reference

- Drag a variable from the Variable Manager onto a data input pin to create an input reference, or to a data output pin to create an output reference.
An input reference performs the same as a **Get** variable node and retrieves the value of the variable at the time of node execution.

An output reference performs the same as a **Set** variable node and assigns the output of that slot to the specified variable at the time of node execution.

**Note**
Changing the name of the variable is still supported when you have variable references to that variable. However, you might notice that the display of the new variable name is not immediately reflected in nodes containing the reference. To refresh the display of the node's variable name references, move over the node with your pointer or close and re-open Script Canvas.

**To select a different variable reference**

1. Choose the gear button next to the variable name on the data pin.
2. Select a different variable from the list.

**To convert a variable reference back to a value or into a variable node**

- Do one of the following:
  - Double-click on the data pin name to toggle between a reference and a value.
  - Right-click on the data pin and choose **Convert to Value** to restore that data pin as a value.
  - Right-click on the data pin and choose **Convert to Variable Node** to create a **Get** variable node from a data input pin, or a **Set** variable node from a data output pin.
To convert a data pin into a variable reference

1. Do one of the following:
   - Double-click on the data pin name to toggle between a value and a reference.
   - Right-click on the data pin and choose Convert to Reference.

2. Use the gear button next to the variable name field that appears and select a variable to reference.

   **Tip**
   A quicker alternative for creating a variable reference is to drag a variable from the Variable Manager onto the data pin.

To convert a variable node into a variable reference

- Do one of the following:
  - Right-click on a Get variable node and choose Convert to References. This converts the node into a variable reference on the node following it.

    **Note**
    If the data output from the Get variable node is not connected to another node, the variable node is deleted.
  - Right-click on a Set variable node and choose Convert to References. This converts the node into a variable reference on the node preceding it.
Note
If the data input to the Set variable node is not connected to another node, the variable node is deleted.

Setting Entity Targets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Nodes can contain an entity property. These properties tell the node which entity to affect. By default, many nodes reference Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can reference a specific entity other than self.

To reference entities for a node

1. From the Node Palette, locate the node that you want to add to your script and drag it to the canvas.
2. In the node, position your pointer over the entity property and then click the target icon.
3. In the Lumberyard Editor viewport or the Entity Outliner, select the entity that you want to reference.
4. To clear the entity, position your pointer over the entity property and then click the x icon.
5. To reset a property back to self, right-double-click the entity property and choose Set to Self.

Deleting Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas offers a variety of techniques for deleting nodes.

To delete one or more nodes and their connections

1. Select the node or nodes that you want to delete.
2. Perform one of the following steps.
   • Press Delete.
   • Right-click, and then choose Delete.

To delete a single node

• On the node that you want to delete, press Alt+Left-click. If the deleted node was connected to one other node, the connection is deleted. If the deleted node was connected to two other nodes, the remaining nodes connect to each other.

Note
You cannot use Alt+Left-click to delete more than one node at a time.

To remove all unused nodes in a graph

• In Script Canvas editor, choose Edit, Remove Unused, Nodes.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas offers a variety of ways to delete connections between nodes.

To delete a connection using the context menu
1. Pause your pointer over the connection. The white line becomes a line of moving blue dashes.
2. Right-click the blue dashed line.
3. In the context menu, choose Delete.

To delete a connection using the Delete key
1. Pause your pointer over the connection. The white line becomes a line of moving blue dashes.
2. Left-click the blue dashed line.
3. Press Delete.

Note
You must left-click the blue dashed line before you press Delete.

To delete connections by using the Alt key
1. With the Alt key pressed, pause your pointer on the line. The line turns red.
2. Left-click the line.
To delete a node's connections by "shaking" the node

1. Using your pointer, select a connected node and move it with a shaking gesture to separate it from the other nodes. If the deleted node was connected to two other nodes, the remaining nodes connect to each other.

2. You can make changes to this option in Edit, Settings, Global Preferences, Shake to Desplice.

Organizing Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas graphs can easily grow in size and complexity and become too big to fit on one screen and still be legible. This presents a challenge to find sections of interest and keep things organized. And when you or others revisit the graph later, it can be difficult to quickly remember or pick up on the logic. To help manage this organizational challenge, the Script Canvas editor provides a handful of tools at your disposal. In the following topics, you will learn how to arrange, comment, and bookmark sections of your graph, and discover productivity-enhancing techniques for quickly selecting multiple nodes and navigating around your graphs.

Topics
- Aligning Nodes (p. 2518)
- Commenting Nodes (p. 2524)
- Grouping Nodes (p. 2526)
- Creating Comment and Group Presets (p. 2530)
- Adding Bookmarks for Script Canvas (p. 2534)
Aligning Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To align selected nodes in a Script Canvas graph, you can use the context menu, hot keys, or the following alignment icons above the graph.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Top align.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Bottom align.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Left align.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Right align.</td>
</tr>
</tbody>
</table>

To align all nodes in a selection to the topmost node

- Use one of the following methods:
  - Click Top Align (ıldır).
  - Press Shift + Up Arrow.
  - Right-click the graph, and choose Align, Align top.

The following animated images show the first two methods.
To align all nodes in a selection to the bottommost node

- Use one of the following methods:
  - Click **Bottom Align** (\(\mathbb{L}\))
  - Press **Shift + Down Arrow**.
  - Right-click the graph, and choose **Align, Align bottom**.
The following animated images show the first two methods.
To align all nodes in a selection to the leftmost node

- Use one of the following methods:
  - Click **Left Align** ( ),
  - Press **Shift + Left Arrow**,
  - Right-click the graph, and choose **Align, Align left**.

The following animated images show the first two methods.
To align all nodes in a selection to the rightmost node

- Use one of the following methods:
  - Click **Right Align** (△).
  - Press **Shift + Right Arrow**.
  - Right-click the graph, and choose **Align, Align right**.

The following animated images show the first two methods.
Commenting Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add comments to your script to describe how it works.

Contents

- Adding Comment Nodes (p. 2524)
- Adding Block Comment Nodes (p. 2524)
- Changing Comment Node Font Settings (p. 2525)

Adding Comment Nodes

A Comment node is a floating block of text that you can move in the Script Canvas editor canvas.

To add a Comment node to your script

1. In the Script Canvas editor, do one of the following:
   - In the Node Palette, click Utilities and then click and drag the Comment node to your script.
   - Right-click your script and choose Add Comment.
2. Double-click the node and then enter a description.
3. To delete a Comment node, do one of the following:
   - Select the node and press Delete.
   - Right-click the node and choose Delete Node.

Adding Block Comment Nodes

The Block Comment node is similar to the Comment node, except that you can also use block comments to contain other nodes and organize your script into sections. You can add other nodes to the Block Comment node to tag a specific region or add colors to organize the different parts of your script.

To add a Block Comment node to your script

1. In the Script Canvas editor, do one of the following:
   - In the Node Palette, click Utilities and then drag the Block Comment node to your script.
   - Right-click the canvas and choose Create Block Comment.
   - If you have a group of nodes together, right-click the canvas and choose Create Block Comment for Selection.
2. For the Block Comment node, double-click the header, and then enter a description.
3. Drag the corners of the node to resize it.
4. Drag your nodes into the **Block Comment** node or expand the **Block Comment** node to contain them.

**Example**

The following **Block Comment** node contains other nodes as a group.

![](image)

5. To move your nodes together, drag the **Block Comment** node header. All nodes inside the **Block Comment** node move together.

6. To delete the node, do one of the following:

   - Select the **Block Comment** node header and press **Delete**.
   - Right-click the header for the **Block Comment** node and choose **Delete Node**.

   **Note**
   
   Nodes inside the **Block Comment** node are not deleted.

**Changing Comment Node Font Settings**

You can change the font settings in comment nodes to label and organize your script. Font settings apply to the entire comment; you cannot specify individual sections of the comment.

**To change the font settings for comment nodes**

1. In the **Script Canvas** editor, choose **View, Node Inspector**.
2. Do one of the following:

   - For a **Comment** node, select the node.
   - For a **Block Comment** node, select the header.
3. In the **Node Inspector**, you can make the following changes:
As your Script Canvas graphs grow in size, you can group nodes to logically organize parts of a script or reduce its visual complexity. Groups can be nested, named, and color-coded.

### Creating and Managing Node Groups

**To create a node group from existing nodes and/or node groups**

1. Select the nodes that you want to group.
2. Right-click the graph and choose **Group** or **New group**.
3. Enter a name for the node group, and then press **Enter**.

**To create an empty group**

- Right-click an empty portion of the graph, and then choose **New group**. You can expand the group's borders and add nodes to it as you would on the main graph.
To collapse a group

- Do one of the following:
  - Double-click the group.
  - Right-click the group, and choose **Collapse**.
  - In **Node Inspector**, select the **Collapse Group** option.

A collapsed group has a border with a dashed line.

To expand a group

- Do one of the following:
  - Double-click the group.
  - Right-click the group, and choose **Expand**.
  - In **Node Inspector**, clear the **Collapse Group** option.

To edit the name of a group

- Do one of the following:
  - Right-click the title bar of the expanded group, and choose **Edit group title**.
  - In **Node Inspector**, in the **Group Name** box, edit the name.

To resize a group's borders to fit the contents of the group

1. To reduce empty horizontal space within the group, double-click the group's left or right border.
2. To reduce empty vertical space within the group, double-click the group's top or bottom border.
To ungroup a group

- Right-click the group, and choose Ungroup.

To delete a group

- Right-click the group, and choose Delete.

Note
Deleting a group removes all of the nodes in the group and their connections to other nodes outside of the group. If you want to remove the group container but keep its nodes and connections, choose Ungroup.

Enabling Groups as Bookmarks

To quickly navigate through the groups in your graphs, you can enable groups as bookmarks.

To bookmark a group

1. Select the group.
2. In Node Inspector, select Enable as Bookmark.

Customizing Groups

You can use the Node Inspector to customize the color of a group and to change the font settings of the group title.

To change the color of a node group

1. Select the node group.
2. In Node Inspector, do one of the following:
   - If you know the RGB values that you want to use, enter them in the Group Color text box.
   - Click the Group Color icon to use the Select Color dialog box.
3. In the Select Color dialog box, specify the color that you want to use. You can choose from basic colors, create a custom color, or click Pick Screen Color to choose a color on your screen with your pointer.
4. Click OK.

To change the font settings for the node group title

1. In Node Inspector, expand Font Settings.
2. Enter or choose the values that you want to use for the group title font. The changes that you make are immediately visible.
Font Setting | Description
--- | ---
**Font Color** | If you know the RGB values that you want to use, enter them in the text box. Otherwise, click the color icon and use the Select Color dialog box to specify a color. The default is black (0,0,0).

**Font Family** | Enter the name of a font family (for example, **Verdana** or **Arial Black**).

**Pixel Size** | Enter a size for the font. Valid values are from 1 through 200. The default is 16.

**Weight** | Choose a preselected font weight. Possible values are Thin, Extra Light, Light, Normal, Medium, Demi-Bold, Bold, or Extra Bold. The default is Normal.

**Style** | Choose a font style. Possible values are Normal, Italic, and Oblique. The default is Normal.

**Vertical Alignment** | Choose the vertical position of the title text within the group header. Possible values are Top, Middle, or Bottom. The default is Top.

**Horizontal Alignment** | Choose the horizontal position of the title text within the group header. Possible values are Left, Center, or Right. The default is Left.

---

### Creating Comment and Group Presets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can save custom node comment and node group settings as presets that you can conveniently reuse. In a team setting, these presets can help you create your own coloring and naming conventions to improve clarity and consistency among Script Canvas graphs.

### Customizing the Look of a Comment or Group

Before you create a preset based on an existing node comment or node group, customize its color and font by using the **Background Color** and **Font Settings** options in **Node Inspector**.
For more information, see Customizing Groups (p. 2528).

Creating and Using Presets

You can create a preset in the Script Canvas editor by using an existing node group or node comment.

To create a preset from a group or comment

1. Right-click the group or comment and choose Create Preset From. The preset that you create saves the font settings and color of the original group or comment. It does not save the display text.

2. In the Set Preset Name dialog box, enter a name for the preset, and then click OK.
3. To use the preset that you created, right-click the canvas, choose **Add Comment** or **Group**, and then choose the preset comment or group that you created.

**Configuring a Default Preset**

You can define a preset as the default for either a node group or a comment. The default group or comment is created when you perform one of the following actions:

- On the Script Canvas editor toolbar, click the new comment icon.
- Press **Ctrl+Alt+M** to create a comment.
- Press **Ctrl+Shift+G** to create a group.

**To configure a default preset**

1. In the Script Canvas editor, choose **Tools, Presets Editor**.
2. For **Construct Type**, choose **Comment** or **Node Group**.
3. For **Is Default**, select the preset that you want to make the default, and then click **OK**.
Removing a Preset

To remove presets, use the Script Canvas Presets Editor.

**To remove a preset**

1. In the Script Canvas editor, choose Tools, Presets Editor.
2. For Construct Type, choose Comment or Node Group.
3. Select the preset that you want to delete.
4. Click Remove, and then click OK.
Notes

- After you configure a preset, you cannot modify it. Use the Presets Editor to remove the preset. Then recreate the preset.
- If you recreate a preset, the changes that you make do not propagate to comments or groups that you created with the earlier version of the preset.

Adding Bookmarks for Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

A bookmark is a shortcut that you can add to a specific location on your Script Canvas graph. You can then use keyboard shortcuts to move quickly to that location. For example, you may have a complex script for a massive character controller that includes sections for movement, camera, and each attack. You can add a bookmark to each section and then press the keyboard shortcut to move the graph view to the specified section.

To add a bookmark

1. In your script, navigate to the node or view that you want.
2. Press Ctrl+1 (or 2 to 9) to add a bookmark. A colored icon appears on the script.

To snap the graph view to a bookmark

- Press 1 to 9 to snap the graph view to the bookmarked location.

To manage your bookmarks

1. In the Script Canvas editor, choose View, Bookmarks.
2. In the Bookmarks window, you can do the following:
• Search for the bookmark name.
• Double-click a bookmark and then enter a new name.
• Click Create to add a bookmark at your current location.
• Select a bookmark and then click Delete to remove it.
• Assign up to nine keyboard shortcuts for your bookmarks. To move to a bookmark that does not have an assigned keyboard shortcut, click the bookmark in the Bookmarks pane.
• Double-click a shortcut and then select a new number.

Note
If you specify a number for a shortcut that a previous shortcut already uses (for example, 5), a dialog box appears. If you want to replace the previous shortcut, click Yes.

Navigating Graphs

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you want to visit all nodes of the same type on a graph – for example, to check their configuration – use the following procedure.

To cycle through instances of the same node
1. In the Node Palette search box, enter the name of the node.
2. In the list of results, click the name of the node whose instances you want to find.
3. To scroll to the right, starting at the leftmost node occurrence, press F8. The relevant nodes are highlighted with an orange glow.
4. To scroll to the left, starting at the rightmost node occurrence, press F7.
5. To return to the beginning or end of the cycle, press F7 or F8 again.
Selecting Nodes Using Hot Keys

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following hot key combinations to select groups of nodes.

- To select all nodes in a graph, press **Ctrl + A**.
- To select all nodes that are connected to the input pins of the selected node, press **Ctrl+Left Arrow**.
• To select all nodes that are connected to the output pins of the selected node, press Ctrl+Right Arrow.

• To select all nodes that are connected to both the input and output pins of a selected node, press Ctrl + Up Arrow.
Script Canvas Functions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Script Canvas editor enables you to create reusable graphs, called functions. A function is available for use as a node in your Script Canvas graphs. Similar to functions in traditional programming languages, functions in the Script Canvas editor promote code reuse and abstraction. They help simplify your graphs by replacing a group of nodes that perform a specific task with one function node. For example, you could move a series of nodes that perform linear interpolation into a function called Interpolate, and move nodes that perform acceleration clamping into a function called ClampAcceleration. Additionally, if you're using this functionality in multiple graphs, functions make updates easier because you need to make changes only in one place.

Your function appears alongside all the other Script Canvas nodes in the Node Palette, categorized under Global Functions.

When you create a function, you define the input and output variables of that function, and all the nodes in between that produce a result.

Topics
- Creating a Script Canvas Function (p. 2538)
- Using a Script Canvas Function in a Graph (p. 2541)
- Function Example: Linear Interpolation (p. 2541)

Creating a Script Canvas Function

Start creating a new function by choosing File, New Function in the Script Canvas editor.

Alternatively, you can create a new function using the function create button located in the upper right corner of the editor canvas.

Function Entry and Exit Points

Functions require entry and exit points. To create these points, right-click a node's input or output execution slot and select Expose in the context menu. Typically, you create an entry point node from the input execution slot of the first node in your function, and an exit point node from the output execution slot of the last node in your function.
**Note**
To change the names of the execution slots on the function's node, edit the name field on the entry point or exit point node.

Optionally, if you're not certain how you want to connect an entry or exit point to the rest of your function, you can create an entry or exit point node using the toolbar buttons: ![Connect its execution slot later, when you're ready.](connect.png)

**Function Data Parameters**

Functions can also have input and output data parameters. Input parameters are the values that are passed in to the function. Output parameters are the values that are returned by the function. These are both defined as variables in the Variable Manager. A function can also have local variables, which are not exposed on the function's node.
The scope of a variable determines if and where the variable will appear on the function node.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Location on Node</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>(None)</td>
<td>This is a local variable, for use only by the function.</td>
</tr>
<tr>
<td>In</td>
<td>Input slot</td>
<td>This is an input parameter. A value for this variable is passed in to the function.</td>
</tr>
</tbody>
</table>
### Using a Script Canvas Function in a Graph

Once a Script Canvas function has been saved, it automatically shows up in the **Node Palette**, under **Global Functions**. The default directory for new functions is your project's `scriptcanvas\functions` directory. If you save them in a subdirectory, or save them under a different project directory, the directory structure is used to categorize the functions within **Global Functions**.

Using a function is just like using any other node in Script Canvas. Simply drag and drop the function into a graph. When it's dropped, a node that represents the function is displayed on the canvas.

#### Function Example: Linear Interpolation

In this example, we create a linear interpolation function. This function is represented by the formula:

\[
\text{Result} = \text{Start} + \text{Time} \times (\text{End} - \text{Start})
\]

In this example you will learn how to do the following:

- Create entry and exit points for a function
- Create input parameter slots
- Create return values

The final function graph should look like this:
1. Start a new function using File, New Function or by using the function create button in the upper right corner of the canvas.

2. Create the following four variables in Variable Manager:
   - Start
   - End
   - Time
   - Result

3. Set the scope of Start, End, and Time to In. These are now your input parameters.

4. Set the scope of Result to Out. This is now your return value.

5. Place the Subtract, Multiply, and Add nodes onto your graph and connect them as shown in the previous image.

6. Using variable references (p. 2511), do the following:
   1. In the Subtract node, reference the End and Start variables, so that Start is subtracted from End.
   2. In the Multiply node, use the result of the subtraction and reference the Time variable, so that \((\text{End} - \text{Start})\) is multiplied by Time.
   3. In the Add node, use the result of the multiplication and reference the Start variable, so that these two values are added together. Then add a reference to the Result variable in the result slot, so that the final value is stored in Result.

7. Create entry and exit points for your function.
   1. Right-click on the Subtract node's In slot and choose Expose from the context menu.
   2. (Optional) Rename the entry point node to In, if you want to follow the naming convention for node input execution slots.
   3. Right-click on the Add node's Out slot and choose Expose from the context menu.
   4. (Optional) Rename the exit point node to Out, if you want to follow the naming convention for node output execution slots.

8. Finally, use File, Save to save the function and name it Interpolate. The function is now ready to use in a Script Canvas graph:
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

From this section you can gain a deeper understanding of how Script Canvas works with the rest of Lumberyard, and learn how to author your own custom nodes for use in Script Canvas graphs.

Script Canvas and the Behavior Context

You can use Script Canvas to expose runtime code in a visual authoring environment using the behavior context. In addition to this topic, it is recommended that you also read the Programmer’s Guide to Entities and Components (p. 973) and Behavior Context (p. 1053).

The following code architecture diagram shows the relationship between Script Canvas and the behavior context in Amazon Lumberyard.
The core Script Canvas code is built as a static library that is linked into the dependent gem and editor gem. This allows the code footprint at run time to be as small as the minimum required to run a Script Canvas graph. It also allows the Script Canvas Editor Gem to contain all the code required to author and develop Script Canvas graphs.

Script Canvas is designed to use behavior context reflection and provide access to the same objects and EBus (p. 1851) that are available in Lua. When you use the behavior context, you do not need to write any Script Canvas-specific code. However, it is important that the way in which your code is reflected to the behavior context remains intuitive in a visual scripting environment.

The behavior context for Script Canvas includes the following benefits:

- Functionality exposed through the behavior context is also available in Lua (and any other scripting solution bound to the behavior context).
- The EBus system makes it easier for you to use decoupled, event-driven programming paradigms.
- Script Canvas can use any functionality that is exposed to the behavior context, even if it comes from other gems. Therefore, when code is reflected to the behavior context, gems can enhance Script Canvas.
- Reflecting gems to the behavior context removes any need to add gem dependencies to Script Canvas.

The Light Component and Script Canvas

So that you can better understand the relationship between the behavior context and Script Canvas, this section discusses the fairly simple Light component. The example shows how its behavior context reflection translates into Script Canvas nodes.

You can use the Light component to give an entity a light. You can configure the light by setting parameters such as color, intensity, and radius. The Light component can also be turned on or off, and you can respond to these events when they occur.
Communication with an entity's Light component is done through two EBuses: LightComponentRequestBus and LightComponentNotificationBus.

A request bus provides methods that can be called on an entity. If an entity has a Light component, the Light component implements the behavior of the requests that are made to it.

Some of the requests that you can issue on a Light Component are the following. For the source code, see the file dev\Gems\LmbrCentral\Code\include\LmbrCentral\Rendering\LightComponentBus.h.

```cpp
//! Turns light on. Returns true if the light was successfully turned on
virtual bool TurnOnLight() { return false; }

//! Turns light off. Returns true if the light was successfully turned off
virtual bool TurnOffLight() { return false; }

//! Toggles light state.
virtual void ToggleLight() {}
```

These requests are part of the LightComponentRequestBus. Their behavior is implemented by LightComponent.

To make these requests accessible for scripting, they must be reflected to the behavior context. This is done in LightComponent::Reflect. The source code is in the file dev\Gems\LmbrCentral\Code\Source\Rendering\LightComponent.cpp.

```cpp
behaviorContext->EBus<LightComponentRequestBus>("Light", "LightComponentRequestBus")
    ->Attribute(AZ::Script::Attributes::Category, "Rendering")
    ->Event("TurnOn", &LightComponentRequestBus::Events::TurnOnLight, "TurnOnLight")
    ->Event("TurnOff", &LightComponentRequestBus::Events::TurnOffLight, "TurnOffLight")
    ->Event("Toggle", &LightComponentRequestBus::Events::ToggleLight, "ToggleLight")
```

When Script Canvas examines the behavior context, it finds these bindings and automatically generates the corresponding nodes for you.
You use EBuses to communicate with an entity's components. To do so, you need an address. All component EBuses derive from `AZ::ComponentBus`, which is addressable by an ID of the type `AZ::EntityId`. For this reason, all nodes from a component EBus have an entry slot for an EntityID. The presence of `Self` in the EntityID field refers to the EntityID of the entity that owns the Script Canvas graph. However, this ID can be assigned to another entity, or even changed to an invalid entity ID.
The other bus that the Light component reflects to the behavior context is `LightComponentNotificationBus`. The following source code is also in the file `dev\Gems\LmbrCentral\Code\Source\Rendering\LightComponent.cpp`.

```cpp
behaviorContext->EBus<LightComponentNotificationBus>("LightNotification",
    "LightComponentNotificationBus", "Notifications for the Light Components")
    ->Attribute(AZ::Script::Attributes::Category, "Rendering")
    ->Handler<BehaviorLightComponentNotificationBusHandler>();
```

Notification buses are also known as **event handlers**. You can use these event handlers on a component to respond to the events that happen to the component. As part of reflection to the behavior context, the preceding code specifies that the `BehaviorLightComponentNotificationBusHandler` handles events for the Light component.

The following code shows the binding for the `BehaviorLightComponentNotificationBusHandler` and defines two events: `LightTurnedOn` and `LightTurnedOff`.

```cpp
class BehaviorLightComponentNotificationBusHandler : public LightComponentNotificationBus::Handler, public AZ::BehaviorEBusHandler
{ public:
    AZ_EBUS_BEHAVIOR_BINDER(BehaviorLightComponentNotificationBusHandler, 
        "{969C5B17-10D1-41DB-8123-6664FA64B4E9}", AZ::SystemAllocator,
        LightTurnedOn, LightTurnedOff);
    // Sent when the light is turned on.
    void LightTurnedOn() override
    {
        Call(FN_LightTurnedOn);
    }
    // Sent when the light is turned off.
    void LightTurnedOff() override
    {
        Call(FN_LightTurnedOff);
    }
};
```

Script Canvas has a node that gives you access to all the events for EBus. You can handle the events that you are interested in.
Objects: The PhysicsComponent Example

Objects that are reflected to the behavior context are also available in Script Canvas. Objects become available in the form of variables.

The Light component does not provide any objects, but the PhysicsComponent provides an example of an object reflected to the behavior context. The Collision class is reflected as a behavior context object that you can access through the use of variables in Script Canvas. You can find the following source code in the file `dev\Gems\LmbrCentral\Code\Source\Physics\PhysicsComponent.cpp`.

```cpp
using Collision = PhysicsComponentNotifications::Collision;
// Information about a collision event
behaviorContext->Class<Collision>();
  ->Attribute(AZ::Script::Attributes::Storage, AZ::Script::Attributes::StorageType::Value)
  ->Property("entity", BehaviorValueProperty(&Collision::m_entity))
  ->Property("position", BehaviorValueProperty(&Collision::m_position))
  ->Property("normal", BehaviorValueProperty(&Collision::m_normal))
  ->Property("impulse", BehaviorValueProperty(&Collision::m_impulse))
  ->Property("velocities", BehaviorValueGetter(&Collision::m_velocities), nullptr)
  ->Property("masses", BehaviorValueGetter(&Collision::m_masses), nullptr)
  ->Property("surfaces", BehaviorValueGetter(&Collision::m_surfaces), nullptr)
;
```
**Note**
During the preview release of Script Canvas, properties must provide both getters and setters to be accessible on a Script Canvas node. Containers such as vectors are currently not supported. For this reason, velocities, masses, and surfaces do not provide a setter.

Most object variables are set as a result of an event. In the case of the preceding Collision example, the Collision variable is returned by the OnCollision event.

You can use the following graph to set the collision variable:
Displaying EBus Event Parameter Names in Script Canvas Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To display parameter names correctly for your EBus events, ensure that you specify custom names when you reflect your events to the behavior context.

If you do not specify names for the parameters, they are given default display names like "1", "2", or "3", as in the following image:
The following code produced the event node in the image:

```cpp
if (auto behaviorContext = azrtti_cast < AZ::BehaviorContext * >(reflectContext))
{
    behaviorContext->EBus < MyBus >("MyBus")
    // This is the category that appears in the Node Palette window
    ->Attribute (AZ::Script::Attributes::Category,
                 "Rendering") ->Event ("SomeEvent",
                           &MyBus::Events::SomeEvent);
}
```

The following version of the same code adds the parameter names `FirstParam` and `SecondParam` and corresponding tooltip text to the `Event` function:

```cpp
if (auto behaviorContext = azrtti_cast<AZ::BehaviorContext*>(reflectContext))
{
    behaviorContext->EBus<MyBus>("MyBus")
    // This is the category that appears in the Node Palette window
    ->Attribute(AZ::Script::Attributes::Category, "Rendering")
    ->Event("SomeEvent", &MyBus::Events::SomeEvent, { { "FirstParam", "First Param Tooltip" }, { "SecondParam", "Second Param Tooltip" } });
}
```

In the node palette window, the parameter names appear as specified:

**Alternate Syntax**

You can also use the following alternate syntax to create parameter override instances before passing them to the `Event` function:
if (auto behaviorContext = azrtti_cast<AZ::BehaviorContext*>(reflectContext))
{
    AZ::BehaviorParameterOverrides someEventParam1 = { "FirstParam", "First Param Tooltip" };
    AZ::BehaviorParameterOverrides someEventParam2 = { "SecondParam", "Second Param Tooltip" };
    behaviorContext->EBus<MyBus>("MyBus")
        // This is the category that appears in the Node Palette window
        ->Attribute(AZ::Script::Attributes::Category, "Rendering")
        ->Event("SomeEvent", &MyBus::Events::SomeEvent, { {someEventParam1,
            someEventParam2} });
}  

Common Programming Problems

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following are some common problems that occur when programming with Script Canvas and the behavior context.

I reflected my class to the behavior context, but it doesn't appear in Script Canvas.

Both the serialization context and the behavior context use the same Reflect function:

```
Reflect(AZ::ReflectContext*)
```

A common mistake is to not keep the reflection scopes separate. For example, you might mistakenly place the BehaviorContext reflection within the SerializeContext scope. The following code examples show the problem and the solution.

Problem

```
void Example::Reflect(AZ::ReflectContext* context)
{
    if (AZ::SerializeContext* serializeContext = azrtti_cast<AZ::SerializeContext*>(context))
    {
        serializeContext->Class<Example>()
            ->Version(1)
            ;
        // Problem! BehaviorContext is inside the SerializeContext scope and will not get reflected.
        if (AZ::BehaviorContext* behaviorContext = azrtti_cast<AZ::BehaviorContext*>(context))
        {
            behaviorContext->Class<Example>()
                ->Method("IsValid", &Example::IsValid)
                ;
        }
    }
}
```

Solution

```
void Example::Reflect(AZ::ReflectContext* context)
```
I exposed a new EBus to the Behavior Context, but my handler is not getting called.

For example, you created an EBus handler and properly exposed it script. Yet, when you try to receive the event in Script Canvas, it does not get triggered.

This is caused by an oversight that is easy to make: newly implemented EBus handlers must be connected before they can receive events. The solution is to ensure that your component connects to the bus, as in the following example:

```cpp
MyBus::BusConnect()
```

Depending on the type of bus, you might have to specify an ID to connect to. For more information, see Working with the Event Bus (EBus) system (p. 1851).

## Creating Custom Nodes in Script Canvas

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Script Canvas uses the behavior context (p. 1053). The preferred way to expose functionality in Script Canvas is through behavior context bindings. Behavior context bindings make functionality available to any scripting solution (such as Lua) that also uses the behavior context. However, if you want to create Script Canvas–specific functionality or customized visual scripting experiences, you can use the Script Canvas custom node API.

If your gem provides custom Script Canvas nodes, you must specify a dependency on the Script Canvas Gem and configure Waf appropriately to compile the nodes. Functionality that you reflect through the behavior context requires no dependency on the Script Canvas Gem.

Custom nodes in Script Canvas take advantage of Automating boilerplate with AZ Code Generator (p. 1869) and boilerplate code. This reduces complexity and lets you focus on implementing the behavior of the node.

The following diagram shows the AZ Code Generator process for Script Canvas.
In **Step 1**, AZ Code Generator processes `MyNode.h` and looks for code generation tags (source code location: `dev\Gems\ScriptCanvas\Code\Include\ScriptCanvas\CodeGen\CodeGen.h`). After processing, AZ Code Generator creates a file called `MyNode.generated.h`.

**Note**
You must include `MyNode.generated.h` in the `MyNode.h` file so that the preprocessor macro that AZ Code Generator produces is resolved by C++ during Step 3.

In **Step 2**, AZ Code Generator processes `MyNode.h` again, but this time produces a `.cpp` file.

**Note**
You must include `MyNode.generated.cpp` the in `MyNode.cpp` file. This ensures that the generated code can be compiled and has access to all the same `#include` declarations as those found in `MyNode.cpp`.

In **Step 3**, AZ Code Generator is finished and the build process continues as normal. `MyNode.cpp` is compiled and succeeds provided all the generated files are correct.

**Topics**
- Script Canvas Tags for AZ Code Generator (p. 2554)
- Dynamic Data Slots (p. 2557)
- Extendable Nodes (p. 2558)
- Implementing Node Behavior (p. 2559)
- Creating Custom Script Canvas Nodes in a Gem (p. 2563)
- Node Libraries (p. 2565)
- Node Contracts (p. 2567)

**Script Canvas Tags for AZ Code Generator**

Script Canvas provides a variety of tags that AZ Code Generator uses. These tags can be found in the source code location `dev\Gems\ScriptCanvas\Code\Include\ScriptCanvas\CodeGen\CodeGen.h`. Because the tags are well documented in the code, this guide focuses on showing how to use them rather than on covering each one in detail.
The following example shows the `Delay` node, which uses a variety of different code generation features. The source code files are located in the directory `dev\Gems\ScriptCanvas\Code\Include \ScriptCanvas\Libraries\Time`.

The first tag is `ScriptCanvas_Node`, which is in the `Countdown.h` file. This tag is used as the class declaration of a node and generates the necessary reflection for the node.

```cpp
ScriptCanvas_Node(Countdown,
    ScriptCanvas_Node::Name("Delay")
    ScriptCanvas_Node::Uuid("{FAEADF5A-F7D9-415A-A3E8-F534BD379B9A}")
    ScriptCanvas_Node::Description("Counts down time from a specified value.")
);
```

Note that while internally the class name is `Countdown`, the code specifies that `Delay` be used as the node name in Script Canvas editor. AZ Code Generator uses the `ScriptCanvas_Node` tag to produce the following code in `Countdown.generated.h`:

```cpp
#define AZ_GENERATED_Countdown
public:
    AZ_COMPONENT(Countdown, "{FAEADF5A-F7D9-415A-A3E8-F534BD379B9A}", ScriptCanvas::Node);

    static void Reflect(AZ::ReflectContext* reflection);
    void ConfigureSlots() override;
    bool IsEntryPoint() const override;
    using Node::GetInput;
    friend struct CountdownProperty;
```

The `ScriptCanvas_Node` looks like the following in the `CodeGen.h` file when you compile the project:

```cpp
define ScriptCanvas_Node(ClassName, ...) AZ_JOIN(AZ_GENERATED_, ClassName)
```

When the project is compiled, the preprocessor finds `AZ_GENERATED_Countdown` in the `Countdown.generated.h` file and replaces that macro with the generated code.

**Generating the Node Topology**

After you have declared the node, the next step is to generate the node's topology. The topology can include a variety of tags, as the following table shows.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ScriptCanvas_In</code></td>
<td>Provides a named <code>Input</code> execution slot to the node.</td>
</tr>
<tr>
<td><code>ScriptCanvas_Out</code></td>
<td>Provides a named <code>Output</code> execution slot to the node.</td>
</tr>
<tr>
<td><code>ScriptCanvas_Property</code></td>
<td>This tag must precede a member variable in the class that you want to expose to Script Canvas for editing and scripting. By default, the property is exposed with an <code>Input</code> and <code>Output</code> slot. However, you can use the <code>Input</code> or <code>Output</code> attributes to expose only one or the other.</td>
</tr>
<tr>
<td><code>ScriptCanvas_PropertyWithDefaults</code></td>
<td>Like <code>ScriptCanvas_Property</code>, but specifies default values.</td>
</tr>
<tr>
<td><code>Property</code></td>
<td>Reflects a property to the serialization context that does not need to be an editable property or an input property. For more information, see Serializing &quot;Hidden&quot; Node Properties (p. 2563).</td>
</tr>
<tr>
<td><code>EditProperty</code></td>
<td>Reflects a property to the serialization context and to the <code>EditContext</code> with <code>EditContext</code> attribute support. For more information, see Serializing &quot;Hidden&quot; Node Properties (p. 2563).</td>
</tr>
</tbody>
</table>
Each of these tags has attributes that can be configured. For example, the `Countdown.h (Delay)` node has the following topology:

```cpp
// Inputs
ScriptCanvas_In(ScriptCanvas_In::Name("In", "When signalled, execution is delayed at this node according to the specified properties.")
  ScriptCanvas_In::Contracts({ DisallowReentrantExecutionContract }));
ScriptCanvas_In(ScriptCanvas_In::Name("Reset", "Resets the delay.")
  ScriptCanvas_In::Contracts({ DisallowReentrantExecutionContract }));

// Outputs
ScriptCanvas_Out(ScriptCanvas_Out::Name("Out", "Signalled when the delay reaches zero.")
  ScriptCanvas_Out::Contracts({ DisallowReentrantExecutionContract }));

// Data
ScriptCanvas_Property(float,
  ScriptCanvas_Property::Name("Time", "Amount of time to delay, in seconds")
  ScriptCanvas_Property::Input);
ScriptCanvas_Property(bool,
  ScriptCanvas_Property::Name("Loop", "If true, the delay will restart after triggering the Out slot")
  ScriptCanvas_Property::ChangeNotify(AZ::Edit::PropertyRefreshLevels::EntireTree)
  ScriptCanvas_Property::Input);
ScriptCanvas_Property(float,
  ScriptCanvas_Property::Name("Hold", "Amount of time to wait before restarting, in seconds")
  ScriptCanvas_Property::Visibility(&Countdown::ShowHoldTime)
  ScriptCanvas_Property::Input);
ScriptCanvas_Property(float,
  ScriptCanvas_Property::Name("Elapsed", "The amount of time that has elapsed since the delay began.")
 ScriptCanvas_Property::Visibility(false)
  ScriptCanvas_Property::Output
  ScriptCanvas_Property::OutputStorageSpec
);
```

In the **Script Canvas** editor, the node shows the **Time**, **Loop**, **Hold**, and **Elapsed** properties that were defined:
Dynamic Data Slots

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

There are cases where a single node can function on multiple different input types (for example, the Lerp Between node can function on numbers and vector objects). To reduce the number of unique nodes, you can use dynamic data slots. Dynamic data slots enable a single node to handle a variety of data types. At the same time, they allow restrictions on the data types that can be connected.

You can use the `ScriptCanvas_DynamicDataSlot` tag to add a DynamicDataSlot to any node, as shown in the following example.

```cpp
ScriptCanvas_DynamicDataSlot(ScriptCanvas::DynamicDataType::Value,
    ScriptCanvas::ConnectionType::Output,
    ScriptCanvas::DynamicDataSlot::Name("Step", "The value of the current step of the lerp.")
    ScriptCanvas::DynamicDataSlot::DynamicGroup("LerpGroup")
ScriptCanvas::DynamicDataSlot::RestrictedTypeContractTag({ Data::Type::Number(),
    Data::Type::Vector2(), Data::Type::Vector3(), Data::Type::Vector4() })
```

The `ScriptCanvas_DynamicDataSlot` tag includes the following code gen attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamicDataType</td>
<td>Allows the user to specify a macro category of dynamic typing information. DynamicDataType has the following supported values:</td>
</tr>
<tr>
<td></td>
<td>• Container – Maps or arrays of any data type.</td>
</tr>
<tr>
<td></td>
<td>• Value – Any non-map or non-array value.</td>
</tr>
<tr>
<td></td>
<td>• Any – Any Container or Value type.</td>
</tr>
<tr>
<td>DynamicGroup</td>
<td>Unifies a group of dynamically typed slots. When one slot has a type, all slots in the group share the same type. This attribute is useful for ensuring that passsthrough values or operands all share a common type.</td>
</tr>
<tr>
<td>RestrictedTypeContractTag</td>
<td>Restricts the data types that the dynamically typed slot accepts. Takes an argument that is a list of supported data types.</td>
</tr>
</tbody>
</table>

Important Notes about Groups

For the purposes of contract restrictions, the grouped slots act as a single unit. If one slot doesn't allow a particular type, no slots in that group allow that type. This behavior is because connection types are propagated through every member of a group and every group connected to that group. This also applies to dynamic data slots that are connected to each other and any dynamic data slots that are connected but ungrouped.

Chained Dynamic Types

When dynamic types are connected, the receiving slot takes on the data type and restrictions of the connecting slot.
Example

Like dynamic groups, any pair of dynamically typed nodes that are linked share the same restrictions. This includes any restrictions from the node group. Additionally, the slots are usually in the most unrestricted state as possible. Unless a display type gives a set of dynamically typed slots a type, the slots remain untyped.

Adding Dynamic Data Slots Programmatically

You can also add dynamic slots programmatically. Use `DynamicDataSlotConfiguration` to define the slot, and then the generic `AddSlot` method to add it, as shown in the following example.

```cpp
SlotId MyNode::AddDynamicSlot(AZStd::string_view name, AZStd::string_view toolTip,
                              ConnectionType connectionType)
{
    DynamicDataSlotConfiguration slotConfiguration;

    // Generic Slot Configuration
    slotConfiguration.m_name = name;
    slotConfiguration.m_toolTip = toolTip;
    slotConfiguration.SetConnectionType(connectionType);
    slotConfiguration.m_addUniqueSlotByNameAndType = false;

    // Contract Descs provides a list of contracts that must be satisfied for a connection to be accepted to this slot.
    //slotConfiguration.m_contractDescs.push_back(TypeRestriction);

    // DynamicDataSlot Specific Configurations
    slotConfiguration.m_dynamicGroup = "DynamicDataGroup";
    slotConfiguration.m_dynamicDataType = DynamicDataType::Value;
    //

    return AddSlot(slotConfiguration);
}
```

For information about `DynamicGroup` and `DynamicDataType`, see the `ScriptCanvas_DynamicDataSlot` attributes table earlier in this topic.

Extendable Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Certain nodes, such as the math operators, can function on an arbitrarily long list of parameters. To maintain this feature without creating a needlessly large node, extendable slots let users add as many slots as they require to a node.

The following code example shows an extendable node interface.

```cpp
// Returns whether or not the specified SlotId should be allowed to be removed from the
given node.
// This should return false, unless the slot was added dynamically through the extender
system.
virtual bool CanDeleteSlot(const SlotId& slotId) const;

virtual bool IsNodeExtendable() const;
virtual int GetNumberOfExtensions() const;

// At display time, the extendable configuration will be enumerated on for display purposes
within GraphCanvas.
// This should return the configuration that provides details about the information
necessary to display and operate
// the ExtenderSlot from graph canvas.
virtual ExtendableSlotConfiguration GetExtensionConfiguration(int extensionCount) const;

// Signals to the node to add a slot for the specific extension. Should return the SlotId
of that slot that was added
// for the specific ExtensionId.
virtual SlotId HandleExtension(AZ::Crc32 extensionId);
```

The following code example shows how to configure an extendable slot.

```cpp
struct ExtendableSlotConfiguration
{
public:
    AZ_TYPE_INFO(ExtendableSlotConfiguration, "{3EA2D6DB-1B8F-451B-A6CE-D5779E56F4A8");
    AZ_CLASS_ALLOCATOR(ExtendableSlotConfiguration, AZ::SystemAllocator, 0);
    ExtendableSlotConfiguration() = default;
    ~ExtendableSlotConfiguration() = default;

    // The name and tooltip to appear on the corresponding GraphCanvas ExtenderSlot.
    AZStd::string m_name;
    AZStd::string m_tooltip;

    // How to group the ExtenderSlot visually on the node.
    AZStd::string m_displayGroup;

    // The ExtensionId used to identifier the individual ExtenderSlots on a node from one
another in the case of
    // multiple ExtenderIds.
    AZ::Crc32 m_identifier;

    // Controls whether the GraphCanvas ExtenderSlot appears as an input pin or an output
    pin.
    ConnectionType m_connectionType = ConnectionType::Unknown;
};
```

Implementing Node Behavior

After you have defined the topology for the node, you can focus on implementing the behavior for the node.
Adding Slots

In addition to using AZ Code Generator to set up the node's topology, it is also possible to manually add execution and/or data slots if preferred. The following example from the IsNull logic node adds a Reference input slot for data reference and an Is Null output slot. The output slot returns a Boolean value depending on the evaluation of the data that was input. The source code is in the location `dev\Gems\ScriptCanvas\Code\Include\ScriptCanvas\Libraries\Logic\IsNull.cpp`.

```cpp
void IsNull::OnInit()
{
    AZStd::vector<ContractDescriptor> contracts;
    auto func = []() { return aznew IsReferenceTypeContract(); };  
    ContractDescriptor descriptor{ AZStd::move(func) }; 
    contracts.emplace_back(descriptor); 
    AddInputDatumUntypedSlot("Reference", &contracts); 
    AddOutputTypeSlot("Is Null", "", AZStd::move(Data::Type::Boolean()), OutputStorage::Optional); 
}
```

When the node is initialized, the added slots appear in the Script Canvas editor.

Receiving Input Signals

Nodes can receive input signals when a node's execution slot is triggered. To detect which signal has been triggered, implement `OnInputSignal`, as in the following example.

```cpp
void OnInputSignal(const SlotIds) override;
```

To get the ID of the input signal, AZ Code Generator provides some convenient helper functions in a namespace that corresponds to the generated node. These helper functions make it easier to access the node's properties and slot IDs.

As an example, in the Delay node two input slots can be signaled: In and Reset.
In the following code, the Delay node's In and Reset slots use the generated helper functions CountdownProperty::GetInSlotId(this); and CountdownProperty::GetResetSlotId(this); to get their IDs. The source code is from dev\Gems\ScriptCanvas\Code\Include\ScriptCanvas\Libraries\Time\Countdown.cpp.

```cpp
void Countdown::OnInputSignal(const SlotId& slot)
{
    const SlotId& inSlotId = CountdownProperty::GetInSlotId(this);
    const SlotId& resetSlotId = CountdownProperty::GetResetSlotId(this);
    if (slot == resetSlotId || (slot == inSlotId && !AZ::TickBus::Handler::BusIsConnected()))
    {
        // Disconnect required when resetting
        AZ::TickBus::Handler::BusDisconnect();
        m_countdownSeconds = CountdownProperty::GetTime(this);
        m_looping = CountdownProperty::GetLoop(this);
        m_holdTime = CountdownProperty::GetHold(this);
        m_currentTime = m_countdownSeconds;
        AZ::TickBus::Handler::BusConnect();
    }
}
```

You can use these IDs to determine what action the node should take.

**Sending Output Signals**

After the Delay node is finished, it uses the SignalOutput(outSlot) function to signal the output slot that execution is ready to continue.

To call the function, you must know the output slot ID to pass. The following example is from Duration.cpp.
const SlotId doneSlot = DurationProperty::GetDoneSlotId(this);
[...]
SignalOutput(doneSlot);

**Note**
If your node is connected to one or more buses during its lifetime, ensure that it disconnects from those buses before it exits. Otherwise, your node might be handling events that it no longer should.

### Querying Inbound Data

The **Delay** node example has the inbound data slots **Time**, **Loop**, and **Hold**.

![Delay node example](image)

Script Canvas properties that specify the `ScriptCanvas_Property::Input` attribute can be queried by the node to do necessary processing. To do this, use the code-generated convenience helpers `CountdownProperty::Get(PropertyName)(this)`.

```cpp
m_countdownSeconds = CountdownProperty::GetTime(this);
m_looping = CountdownProperty::GetLoop(this);
m_holdTime = CountdownProperty::GetHold(this);
```

You can often use these properties on the stack; you do not have to assign these properties to member variables. In the **Delay** node example, member variables are used to cache the values.

### Sending Outbound Data

Many nodes might want to return a value or push forward data as a result of a computation. In the **Delay** node example, the **Elapsed** slot outputs the elapsed time.

![Delay node example](image)
To output the elapsed time, the node gets the ID of the Elapsed slot and then pushes a data value into it (dev\Gems\ScriptCanvas\Code\Include\ScriptCanvas\Libraries\Time\Countdown.cpp).

```cpp
const SlotId elapsedSlot = CountdownProperty::GetElapsedSlotId(this);
Datum o(Data::Type::Number(), Datum::eOriginality::Copy);
o.Set(m_elapsedTime);
if (auto* slot = GetSlot(elapsedSlot))
{
    PushOutput(o, *slot);
}
```

### Serializing "Hidden" Node Properties

In some cases you might want your node to serialize its properties but not expose them as slots on a node. In this case, the Property and EditProperty tags are useful.

You can use the Property tag to serialize any property without exposing it to the node's property grid. Because it is not exposed, it is not user configurable.

The EditProperty tag both serializes and provides an EditContext for the specified property. This makes the property user configurable through the node's property grid.

### Creating Custom Script Canvas Nodes in a Gem

To have a gem support custom Script Canvas nodes, you must configure it to take a dependency on Script Canvas and then configure Waf. The following procedure uses the Starting Point Input Gem as an example.

**To create a Script Canvas dependency for a gem**

1. Open the gem's gem.json file. (For reference, see the source at \dev\Gems\StartingPointInput\gem.json)

```json
{
    "Dependencies": [,
    {
        "Uuid": "59b1b2acc1974aae9f18faddcaddac5b",
        "VersionConstraints": [
            "~>0.1"
        ],
        "_comment": "InputManagementFramework"
    },
    {
        "Uuid": "869a0d0ec11a45c299917d45c81555e6",
        "VersionConstraints": [">
```
2. In the Dependencies key at the beginning of the file, add the following entry for Script Canvas:

```json
{
    "Uuid" : "869a0d0ec11a45c299917d45c81555e6",
    "VersionConstraints" : [">=0.1.0"],
    "_comment" : "ScriptCanvas"
}
```

**Note**
To use this dependency, the Script Canvas Gem must be enabled in the Project Configurator when your gem is enabled.

3. Configure Waf to find the path to the AzCodeGeneration driver. To see how to do this, you can follow the example of the Starting Point Input Gem's .wscript file.

The following is the full text of the .wscript file. Substeps are provided after the full text.

```python
def build(bld):
    # Need to build a /dev relative path; otherwise, a gem/code relative path is used.
    import os
    driver_node = bld.path.find_or_declare('Gems/ScriptCanvas/Code/Include/ScriptCanvas/
    gem_node = bld.path.find_or_declare('Gems/StartingPointInput/Code/
    script_canvas_dir = driver_node.path_from(gem_node)
    AZ_CODEGEN_ARGS = az_code_gen = [{
        'files'   : ['Source/InputNode.h'],
        'scripts' : [os.path.join(script_canvas_dir, 'CodeGen/Drivers/ScriptCanvasNode.py')],
        'arguments' : [
#'-OnlyRunDiagnosticsOnMainFile=true',
#'-SuppressDiagnostics=false',
#'-SuppressErrorsAsWarnings=false',
#'-output-redirection=file',
#'-SuppressIncludeNotFoundError=false',
],
        'options' : ['PrintOutputRedirectionFile']
    }]
    bld.DefineGem(
    use    = [ 'AzFramework', 'ScriptCanvas' ],
    includes = [ bld.Path('Code/CryEngine/CryAction'),
                 bld.Path('Code/CryEngine')
                ],
    export_includes = [bld.Path('Code/CryEngine')],
    # Suppressing level 1 warning C4351 on windows
    # new behavior: elements of array 'array' will be default initialized
    win_cxxflags = ['/wd4351'],
    features = ['az_code_gen'],
    win_defines = [],
    file_list = [ 'startingpointinput.waf_files' ],
    test_all_file_list = [ 'startingpointinput_tests.waf_files' ],
    az_code_gen = AZ_CODEGEN_ARGS,
    editor = dict(
        az_code_gen = AZ_CODEGEN_ARGS,
        file_list = [ 'startingpointinput_editor.waf_files' ],
    )
```

Version 1.28
2564
a. In the first section of the .wscript file, tell AZ Code Generator where to find the Script Canvas drivers and templates. This section is required.

```python
import os
driver_node = bld.path.find_or_declare('Gems/ScriptCanvas/Code/Include/ScriptCanvas/')
gem_node = bld.path.find_or_declare('Gems/StartingPointInput/Code/')
script_canvas_dir = driver_node.path_from(gem_node)
```

b. In the next section, create a configuration object that specifies the files that you want AZ Code Generator to process. For build performance, AZ Code Generator runs only on the files that you specify in the wscript file. Therefore you must add to the `files` section the source files for any new node on which you want to use AZ Code Generator:

```python
AZ_CODEGEN_ARGS = az_code_gen = [{
    'files'   : ['Source/InputNode.h'],
    'scripts' : [os.path.join(script_canvas_dir, 'CodeGen/Drivers/ScriptCanvasNode.py')],
}
# Note: Only the files specified in the above 'files' section will be run through AZ Code Generator!
```

c. Configure the gem to use AZ Code Generator.

```python
use             = [ 'AzFramework', 'ScriptCanvas' ],  # Ensures that the core ScriptCanvas code is statically linked.
features        = ['az_code_gen'], # Enables AZ Code Generator for this gem.
az_code_gen     = AZ_CODEGEN_ARGS, # Applies the configuration parameters that you previously defined.
```

d. In the last section, provide configuration for the Editor Gem.

```python
editor = dict(  
az_code_gen = AZ_CODEGEN_ARGS,  
file_list = ['startingpointinputEditor.waf_files'],
)
```

Because Script Canvas is both an edit time tool and a runtime system, it requires that your gem provide an Editor Gem. This ensures that the proper .dll files are loaded with the relevant symbols both when the project is compiled and at run time.

---

**Node Libraries**

Nodes in a gem are typically stored in node libraries. These libraries aid in the registration of the nodes with the Script Canvas Gem. They also help organize the nodes within the Script Canvas editor’s **Node Palette**.

The following procedure uses the Script Canvas Diagnostic Library Gem, which is an example of how to create gems that extend Script Canvas. The source code files are in the directory `\dev\Gems\ScriptCanvasDiagnosticLibrary\Code\Source`.

**To create a node library for a gem with Script Canvas**

1. Declare your library. The following example (Debug.h) shows a library declaration from the Script Canvas Diagnostic Library Gem.
struct Debug : public Library::LibraryDefinition
{
    AZ_RTTI(Debug, "{3E28E41D-F4C9-4542-A08F-2B1F5DAA9509}",
             Library::LibraryDefinition);
    static void Reflect(AZ::ReflectContext*);
    static void InitNodeRegistry(NodeRegistry& nodeRegistry);
    static AZStd::vector<AZ::ComponentDescriptor*> GetComponentDescriptors();
};

2. After you declare the library, reflect it. The following example is from Debug.cpp.

void Debug::Reflect(AZ::ReflectContext* reflection)
{
    AZ::SerializeContext* serializeContext =
        azrtti_cast<AZ::SerializeContext*>(reflection);
    if (serializeContext)
    {
        serializeContext->Class<Debug, Library::LibraryDefinition>()
            ->Version(1);
        AZ::EditContext* editContext = serializeContext->GetEditContext();
        if (editContext)
        {
            editContext->Class<Debug>("Debug","")
                ->ClassElement(AZ::Edit::ClassElements::EditorData,"")
                ->Attribute(AZ::Edit::Attributes::Icon,
                             "Editor/Icons/ScriptCanvas/Debug.png")
        }
    }
}

3. Use the InitNodeRegistry and AddNodeToRegistry functions to register the library's nodes (Debug.cpp).

void Debug::InitNodeRegistry(NodeRegistry& nodeRegistry)
{
    Library::AddNodeToRegistry<Debug, Nodes::Debug::Log>(nodeRegistry);
}
AZStd::vector<AZ::ComponentDescriptor*> Debug::GetComponentDescriptors()
{
    return AZStd::vector<AZ::ComponentDescriptor*>({
        Nodes::Debug::Log::CreateDescriptor()});
}

4. Ensure that your library and its nodes are reflected in your gem's Reflect function.

ScriptCanvas::Libraries::Debug::Reflect(context);

5. Add the following code your gem's Init function. The following example is from ScriptCanvasDiagnosticSystemComponent.cpp. This code is important because it inserts your gem's nodes into the Script Canvas Gem's environment.

AZ::EnvironmentVariable<ScriptCanvas::NodeRegistry> nodeRegistryVariable =
    AZ::Environment::FindVariable<ScriptCanvas::NodeRegistry>(ScriptCanvas::s_nodeRegistryName);
if (nodeRegistryVariable)
{
    ScriptCanvas::NodeRegistry& nodeRegistry = nodeRegistryVariable.Get();
    ScriptCanvas::Libraries::Debug::InitNodeRegistry(nodeRegistry);
6. Make sure the library's descriptor is registered in your gem's module. The following code is from ScriptCanvasDiagnosticLibraryGem.cpp.

```cpp
Module::Module()
    : AZ::Module()
{
    m_descriptors.insert(m_descriptors.end(), {
        ScriptCanvasDiagnostics::SystemComponent::CreateDescriptor(),
    });
    AZStd::vector<AZ::ComponentDescriptor*> componentDescriptors(ScriptCanvas::Libraries::Debug::GetComponentDescriptors());
    m_descriptors.insert(m_descriptors.end(), componentDescriptors.begin(), componentDescriptors.end());
}
```

### Node Contracts

Script Canvas uses contracts to validate which node's slots are permitted to connect to other slots. These contracts have validation rules that prevent runtime connections between invalid configurations. The Script Canvas editor uses these contracts to prevent users from creating invalid operations.

Script Canvas has a built-in set of contracts that are used to validate nodes that are reflected to the behavior context. If you create a custom node, you can choose to create or specify contracts directly on slots.

The `Delay` node has the following example of a contract on a custom node:

```cpp
ScriptCanvas_In(ScriptCanvas_In::Name("In", "When signalled, execution is delayed at this
node in accordance with the specified properties.")
    ScriptCanvas_In::Contracts({ DisallowReentrantExecutionContract }));
```

In this case, the DisallowReentrantExecutionContract is specified for the `In` slot. This means that the node does not permit its `Out` slot to connect directly back into its `In` slot.

Slots can have (and often do have) multiple contracts.

### Script Canvas Node Reference

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This reference includes the various types of Script Canvas editor nodes, their uses, and their inputs and outputs.

**Tip**

On the Node Palette or on the node canvas, you can pause on the node name and the node parameters for more information.

**Topics**

- Rendering Nodes (p. 2568)

Some nodes are hidden from the default Script Canvas editor view.
To show hidden nodes in the Script Canvas editor

1. In Lumberyard Editor, choose **Tools, Script Canvas**.
2. In the **Script Canvas** editor, choose **Edit, Settings, Global Preferences**.
3. In the **Global Preferences** window, choose **Show nodes excluded from preview**.

### Rendering Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

See the following **Rendering** nodes for the **Script Canvas** editor.

**Topics**
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use environment nodes in the Script Canvas to configure sky elements, such as sun position, moon position, and the skybox material. You can use these nodes to have specific control over the individual elements.

Some environment nodes (for example, Set Sun Latitude (p. 2577)) have a Force Update parameter. You can use this parameter to make sudden changes to the sky appearance. Since time normally passes slowly with only gradual changes in sky appearance, Lumberyard updates only a portion of the sky for each frame, for better performance. When the sky appearance changes gradually, players are unlikely to notice this optimization. However, if the sky appearance suddenly changes by a large amount, players can notice unusual effects. To avoid this, specify True for the Force Update parameter to update the entire sky in a single frame.

You can use the time of day system to manage high-level control for automatically updating the appearance of the sky. For more information, see Adding Sky Effects (p. 1265) and Time of Day Nodes (p. 2672).

Topics
- Get Moon Latitude (p. 2570)
- Get Moon Longitude (p. 2570)
- Get Sun Latitude (p. 2571)
- Get Sun Longitude (p. 2571)
- Get Wind Direction (p. 2572)
- Set Moon Latitude (p. 2573)
- Set Moon Longitude (p. 2573)
- Set Skybox Angle (p. 2574)
- Set Skybox Material (p. 2575)
- Set Skybox Stretch (p. 2576)
- Set Sun Latitude (p. 2577)
- Set Sun Longitude (p. 2577)
- Set Wind Direction (p. 2578)
Get Moon Latitude

Returns the moon’s latitude in the sky.

Contents
- Inputs (p. 2570)
- Outputs (p. 2570)

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

Inputs

Get Moon Longitude

Returns the moon’s longitude in the sky.

Contents
- Inputs (p. 2571)
- Outputs (p. 2571)

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Latitude</td>
<td>Number</td>
<td>Current moon latitude.</td>
</tr>
</tbody>
</table>
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Longitude</td>
<td>Number</td>
<td>Current moon longitude.</td>
</tr>
</tbody>
</table>

Get Sun Latitude

Returns the sun's latitude in the sky.

Contents

- Inputs (p. 2571)
- Outputs (p. 2571)

Get Sun Longitude

Returns the sun's longitude in the sky.

Contents
Get Sun Longitude

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Longitude</td>
<td>Number</td>
<td>Current sun longitude.</td>
</tr>
</tbody>
</table>

Get Wind Direction

Returns the direction of the global wind. The length of the Wind Direction vector indicates the strength of the wind.

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Wind Direction</td>
<td>Vector3</td>
<td>Current global wind direction.</td>
</tr>
</tbody>
</table>

Set Moon Latitude

Sets the moon's latitude in the sky.

Contents

- Inputs (p. 2573)
- Outputs (p. 2573)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Latitude</td>
<td>Number</td>
<td>Moon latitude value to specify.</td>
</tr>
<tr>
<td>Force Update</td>
<td>Boolean</td>
<td>Forces an immediate update of the entire sky. Use this parameter to make sudden changes to the moon position. For more information, see Environment Nodes (p. 2569).</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Set Moon Longitude

Sets the moon's longitude in the sky.
Contents

- Inputs (p. 2574)
- Outputs (p. 2574)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Longitude</td>
<td>Number</td>
<td>Moon longitude value to specify.</td>
</tr>
<tr>
<td>Force Update</td>
<td>Boolean</td>
<td>Forces an immediate update of the entire sky. Use this parameter to make sudden changes to the moon position. For more information, see Environment Nodes (p. 2569).</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Set Skybox Angle

Sets the rotation angle for the skybox. This rotates the skybox around the z-axis.

For more information, see Creating Static Skyboxes (p. 1272).

Contents

- Inputs (p. 2575)
- Outputs (p. 2575)
### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Angle</td>
<td>Number</td>
<td>Rotation angles in degrees.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Set Skybox Material

Apply a specified material to the skybox to give the sky a different look. The material must be a skybox material.

For more information, see Creating Static Skyboxes (p. 1272).

### Contents
- Input (p. 2575)
- Outputs (p. 2576)
Lumberyard User Guide
Script Canvas Node Reference

### Pin Type Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material</td>
<td>Material</td>
<td>Material to specify. The material must be a skybox material.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Set Skybox Stretch**

Sets the stretch factor, which affects how the skybox stretches vertically. Stretching the skybox lowers the horizon line.

**Contents**
- Inputs (p. 2576)
- Outputs (p. 2576)

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Vertical stretch factor. A value of 0 does not stretch the skybox. Specify a higher value to lower the horizon line. Default value: 0</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Set Sun Latitude

Sets the sun's latitude in the sky.

Contents

- Inputs (p. 2577)
- Outputs (p. 2577)

## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Latitude</td>
<td>Number</td>
<td>Sun latitude value to specify.</td>
</tr>
<tr>
<td>Force Update</td>
<td>Boolean</td>
<td>Forces an immediate update of the entire sky. Use this parameter to make sudden changes to the sun position. For more information, see Environment Nodes (p. 2569).</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Set Sun Longitude

Sets the sun's longitude in the sky.

Contents

- Inputs (p. 2578)
- Outputs (p. 2578)
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Longitude</td>
<td>Number</td>
<td>Sun longitude value to specify.</td>
</tr>
<tr>
<td>Force Update</td>
<td>Boolean</td>
<td>Forces an immediate update of the entire sky. Use this input to make sudden changes to the sun position. For more information, see Environment Nodes (p. 2569).</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node finishes.</td>
</tr>
</tbody>
</table>

**Set Wind Direction**

Sets the direction of the global wind. The length of the Wind Direction vector indicates the strength of the wind.

**Contents**
- Inputs (p. 2579)
- Outputs (p. 2579)
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Wind Direction</td>
<td>Vector3</td>
<td>Wind direction to specify.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Material Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the **Script Canvas** editor to interact with materials in two ways. The following section describes how to access and manipulate materials directly, which affects all entities to which the material is applied. For example, you can load materials, get and set parameters, and apply the material to entities. Changes made to a material affect all entities that use that material.

You can also use the **Script Canvas** editor to manipulate materials on a single entity. For more information, see Material Owner Nodes (p. 2591).

**Topics**

- Clone (p. 2579)
- Find by Name (p. 2580)
- Find Sub-Material (p. 2581)
- Get Param Color (p. 2583)
- Get Param Number (p. 2584)
- Get Param Vector3 (p. 2584)
- Get Param Vector4 (p. 2585)
- Load by Name (p. 2586)
- Set Param Color (p. 2587)
- Set Param Number (p. 2588)
- Set Param Vector3 (p. 2589)
- Set Param Vector4 (p. 2590)

**Clone**

Creates a copy of the specified material. The material must already be loaded into memory, so that the node can find the material. If the material is not found, the node returns Invalid.
In some cases, you might need to store the material to use later. For more information, see Material Variables (p. 2676).

**Warning**
Don’t trigger this node for every frame. Each time the In event is received, a copy of the material is created.

**Contents**
- Inputs (p. 2580)
- Outputs (p. 2580)

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to be cloned.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The cloned material.</td>
</tr>
</tbody>
</table>

**Find by Name**

Finds and returns the material with the specified name. The material must be loaded into memory in order to be found.

In some cases, the **Find by Name** node might not find the material as expected.

**Example**

1. You have an entity named *Crate* that has a material file named *crate.mtl*.
2. In your script, you use the **Find by Name (p. 2580)** node to access the *crate.mtl* file. You want to trigger the node when the graph starts with the **On Graph Start** node. However, it's possible that the **Find by Name (p. 2580)** node can't load the material immediately. It might take additional frames before the entity finishes loading the material file.
3. In this example, you can use one of the following nodes to detect when the *Crate* entity finishes loading the *crate.mtl* file:
• Use the **Load by Name (p. 2586)** node instead of the **Find by Name (p. 2580)** node.
• Use the **On Ready (p. 2599)** node instead of the **On Graph Start** node.

In some cases, you might need to store the material for later use. For more information, see [Material Variables (p. 2676)](#).

**Contents**
- Inputs (p. 2581)
- Outputs (p. 2581)

---

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material Name</td>
<td>String</td>
<td>The path name of a material to find. For more information, see Finding the Material Name (p. 2679).</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material that was found. Returns Invalid if the material is not found.</td>
</tr>
</tbody>
</table>

**Find Sub-Material**

Finds and returns the submaterial with the specified name and submaterial ID.

Lumberyard has two types of material assets:

- **Materials** – A basic single item that represents one material.
- **Multimaterials** – A material that contains multiple submaterials inside it.
You can use the **Find Sub-Material** node to access a submaterial that is inside a multimaterial. To access the submaterial, specify the name of the multimaterial and the ID of one of its submaterials. Material IDs range from the number of available submaterials. For example, if a multimaterial has five submaterials, you can specify a value from 1 to 5.

To access a single material, you can use the **Find by Name (p. 2580)** node.

**Contents**

- Inputs (p. 2582)
- Outputs (p. 2582)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material Name</td>
<td>String</td>
<td>The path name of the material to find. The material must be a multimaterial.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see <a href="#">Finding the Material Name</a>.</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>ID of the submaterial in the multimaterial. IDs start at 1.</td>
</tr>
<tr>
<td>Should Load</td>
<td>Boolean</td>
<td>If true, the submaterial is loaded if it is not already available.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
### Get Param Color

Returns a material parameter as a color value.

**Contents**
- Inputs (p. 2583)
- Outputs (p. 2583)

#### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to query.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to return.</td>
</tr>
</tbody>
</table>

For more information, see [Finding the Material Parameter Name](p. 2680).

#### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>The queried value. If the parameter is not found, the value is (0, 0, 0, 0).</td>
</tr>
</tbody>
</table>
Get Param Number

Returns a material parameter as a numerical value.

Contents

- Inputs (p. 2584)
- Outputs (p. 2584)

## Get Param Number

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to query.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to return. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Number</td>
<td>Number</td>
<td>The queried value. If the parameter is not found, the value is 0.</td>
</tr>
</tbody>
</table>

Get Param Vector3

Returns a material parameter as a Vector3 value.

Contents

- Inputs (p. 2585)
- Outputs (p. 2585)
Get Param Vector3

Material

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to query.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to return.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector3</td>
<td>Vector3</td>
<td>The queried value. If the parameter is not found, the value is (0, 0, 0).</td>
</tr>
</tbody>
</table>

Get Param Vector4

Returns a material parameter as a Vector4 value.

Contents

- Inputs (p. 2586)
- Outputs (p. 2586)
Get Param Vector4

Material

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to query.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to return.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
</tbody>
</table>

Out

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector4</td>
<td>Vector4</td>
<td>The queried value. If the parameter is not found, the value is (0, 0, 0, 0).</td>
</tr>
</tbody>
</table>

Load by Name

Finds and returns the material with the specified name. Loads the material if the material is not already loaded.

Contents
- Inputs (p. 2587)
- Outputs (p. 2587)
In some cases, you might need to store the material to use later. For more information, see Material Variables (p. 2676).

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material Name</td>
<td>String</td>
<td>The path name of a material to find and load. For more information, see Finding the Material Name (p. 2679).</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material that was found. Returns Invalid if the material is not found.</td>
</tr>
</tbody>
</table>

**Set Param Color**

Sets a material parameter with a color value.

**Contents**

- Inputs (p. 2588)
- Outputs (p. 2588)
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to modify.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see <a href="#">Finding the Material Parameter Name</a> (p. 2680)</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Set Param Color**

Sets a material parameter with a number value.

**Contents**

- Inputs (p. 2589)
- Outputs (p. 2589)
### Set Param Number

**Material**

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to modify.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to set.</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished</td>
</tr>
</tbody>
</table>

**Set Param Vector3**

Sets a material parameter with a Vector3 value.

**Contents**

- Inputs (p. 2590)
- Outputs (p. 2590)
## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to modify.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Vector3</td>
<td>Vector3</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

### Set Param Vector4

Sets a material parameter with a Vector4 value.

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Contents
- Inputs (p. 2591)
- Outputs (p. 2591)
Set Param Vector4

Material

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The material to modify.</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Vector4</td>
<td>Vector4</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Material Owner Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Script Canvas editor to interact with materials in two ways. The following section describes how to manipulate materials on a single entity. When you apply changes to the entity that owns the material, the entity's material is automatically cloned. The changes don't affect other entities that have the same material.

An entity is considered a material owner if it has a component that supports materials. This includes the Mesh (p. 684) and Decal (p. 585) components. The Actor (p. 536) component is not supported at this time.

You can also access and manipulate materials directly, which affects all entities to which the material is applied. For more information, see Material Nodes (p. 2579).
Important
Depending on the material owner type, the material asset might not be available when the script is first activated (OnActivate in Lua or the On Graph Start node in Script Canvas). This lack of availability can occur when assets are still loading. You can use the On Ready node instead of the On Graph Start node, or use the Is Ready node.

Topics
- Get Material (p. 2592)
- Get Param Color (p. 2593)
- Get Param Number (p. 2594)
- Get Param Vector3 (p. 2595)
- Get Param Vector4 (p. 2597)
- Is Ready (p. 2598)
- On Ready (p. 2599)
- Set Material (p. 2600)
- Set Param Color (p. 2601)
- Set Param Number (p. 2602)
- Set Param Vector3 (p. 2603)
- Set Param Vector4 (p. 2604)

Get Material
Returns an entity's current material.

Contents
- Inputs (p. 2592)
- Outputs (p. 2593)

Get Material

\[
\text{Material Owner}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Pin} & \text{Type} & \text{Description} \\
\hline
\text{In} & \text{Event} & \text{Triggers the node.} \\
\text{Source} & \text{EntityID} & \text{References a specific entity from which events are generated. By default, it references Self, the entity to which the Script} \\
\hline
\end{array}
\]

Note
If you use the Set Material node and specify an invalid material and then call the Get Material node, Get Material returns the entity's default material instead of Invalid.
## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The entity’s current material.</td>
</tr>
</tbody>
</table>

### Get Param Color

Returns a material's color parameter value for the specified entity.

**Contents**
- [Inputs](#inputs) (p. 2593)
- [Outputs](#outputs) (p. 2594)

## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references <strong>Self</strong>, the entity to which the <strong>Script Canvas</strong> component attaches the current script.</td>
</tr>
</tbody>
</table>

---

*Canvas (p. 818) component attaches the current script.*

You can also select another entity. For more information, see Setting Entity Targets (p. 2514).
### Pin | Type | Description
--- | --- | ---
|  |  | You can also select another entity. For more information, see Setting Entity Targets (p. 2514).

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>The queried value. If the parameter is not found, the value is (0, 0, 0, 0).</td>
</tr>
</tbody>
</table>

### Get Param Number

Returns a material's numerical parameter value for the specified entity.

#### Contents
- Inputs (p. 2595)
- Outputs (p. 2595)
**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Number</td>
<td>Number</td>
<td>The queried value. If the parameter is not found, the value is 0.</td>
</tr>
</tbody>
</table>

**Get Param Vector3**

Returns a material's Vector3 parameter value for the specified entity.

**Contents**

- Inputs (p. 2596)
- Outputs (p. 2596)
### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector3</td>
<td>Vector3</td>
<td>The queried value. If the parameter is not found, the value is (0, 0, 0).</td>
</tr>
</tbody>
</table>
Get Param Vector4

Returns a material's Vector4 parameter value for the specified entity.

Contents

- Inputs (p. 2597)
- Outputs (p. 2598)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector4</td>
<td>Vector4</td>
<td>The queried value. If the parameter is not found, the value is (0, 0, 0, 0).</td>
</tr>
</tbody>
</table>

### Is Ready

Indicates whether the material owner finished loading its assets and the material is ready.

### Contents

- Inputs (p. 2598)
- Outputs (p. 2598)

![Is Ready](image)

## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references <strong>Self</strong>, the entity to which the <strong>Script Canvas</strong> (p. 818) component attaches the current script. You can also specify another entity. For more information, see <strong>Setting Entity Targets</strong> (p. 2514).</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
### Pin Type Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>Indicates whether the material owner is ready, so that request functions can be called.</td>
</tr>
</tbody>
</table>

### On Ready

This node is triggered when the material owner finishes loading its assets. We recommend that you use this event to trigger functions for the Get Param* and Set Param* nodes (for example, the Get Param Color (p. 2593) and Set Param Color (p. 2601) nodes). Otherwise, the material asset might not be available when the script is first activated. For example, a Mesh (p. 684) component does not load its mesh until at least one frame after the script starts.

### Contents
- Inputs (p. 2599)
- Outputs (p. 2600)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also specify another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the material owner's material is available for use.</td>
</tr>
</tbody>
</table>

## Set Material

Sets an entity's material. If the material is `Invalid`, this node removes the entity's material override. The entity uses its default material, if it has one.

### Contents
- Inputs (p. 2600)
- Outputs (p. 2600)

![Set Material](image)

## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references <code>Self</code>, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>The new material.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Set Param Color

Sets a material's color parameter value for the specified entity.

Contents
- Inputs (p. 2601)
- Outputs (p. 2602)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to update. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>The new value to apply.</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Set Param Number

Sets a material's numerical parameter for the specified entity.

Contents

- Inputs (p. 2602)
- Outputs (p. 2603)

---

Set Param Number

Material Owner

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to update. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Number</td>
<td>Number</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

---

Version 1.28

2602
### Pin Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Set Param Vector3

Sets a material's Vector3 parameter value for the specified entity.

### Contents
- Inputs (p. 2603)
- Outputs (p. 2604)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
</tbody>
</table>
## Set Param Vector4

Sets a material's Vector4 parameter value for the specified entity.

### Contents
- Inputs (p. 2604)
- Outputs (p. 2605)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param Name</td>
<td>String</td>
<td>The name of the material parameter to update. For more information, see Finding the Material Parameter Name (p. 2680).</td>
</tr>
<tr>
<td>Vector3</td>
<td>Vector3</td>
<td>The new value to apply.</td>
</tr>
<tr>
<td>Material ID</td>
<td>Number</td>
<td>If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1.</td>
</tr>
<tr>
<td>Pin</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, it references <strong>Self</strong>, the entity to which the <strong>Script Canvas</strong> (p. 818) component attaches the current script. You can also select another entity. For more information, see <strong>Setting Entity Targets</strong> (p. 2514).</td>
</tr>
</tbody>
</table>

| Param Name   | String     | The name of the material parameter to update. For more information, see **Finding the Material Parameter Name** (p. 2680). |
| Vector4      | Vector4    | The new value to apply.                                                                         |
| Material ID  | Number     | If the material owner has a multimaterial, use this parameter to select a specific submaterial. IDs start at 1. |

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Post Effects Nodes**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/sdk-for-unity/) or visit the [AWS Game Tech blog](https://aws.amazon.com/en/about-aws/solutions/game-tech/) to learn more.

You can configure the following post effect nodes in the **Script Canvas** editor. Post effects are useful if you want to enhance a scene after it renders, such as color, blur, or water effects. You can enable an effect once and the effect remains active until you disable it. If you want to animate the settings of an effect or transition in or out, you can trigger the **Enable** node for each frame with different settings.

**Important**

To use post effects nodes, you must enable the **Graphics Scripting** (p. 1143) gem for your game project. For more information, see **Enabling Gems** (p. 1064).

**Topics**

- [Apply Effect Group At Position](p. 2606)
- [Disable Blur](p. 2607)
- [Disable Color Correction](p. 2608)
- [Disable Depth of Field](p. 2608)
- [Disable Directional Blur](p. 2609)
- [Disable Effect Group](p. 2610)
Apply Effect Group At Position

Applies an effect group at a specific position in the world. You must use this node instead of the Enable Effect Group (p. 2622) node for effect group xml files that include the fadeDistance parameter.

The strength of the effect depends on the camera's proximity to the specified location. You can call this node multiple times to apply the same effect group at multiple locations. The effect strength is cleared for each frame, so the node needs to be triggered every frame.

For more information, see Setting Effect Strength Based on Camera Distance (p. 1772).

Contents

- Inputs (p. 2607)
- Outputs (p. 2607)
## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Group Name</td>
<td>String</td>
<td>Relative path to the effect group xml file.</td>
</tr>
<tr>
<td>Position</td>
<td>Vector3</td>
<td>World position where the effect is applied.</td>
</tr>
</tbody>
</table>

## Outputs

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the effect starts.</td>
</tr>
</tbody>
</table>

### Disable Blur

Disables the blur filter.

To enable the effect, see Enable Blur (p. 2616).

### Contents

- Inputs (p. 2608)
- Outputs (p. 2608)
## Disable Color Correction

Disables color correction parameters.

To enable the effect, see Enable Color Correction (p. 2617).

### Contents
- Inputs (p. 2608)
- Outputs (p. 2608)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

## Disable Depth of Field

Disables the depth of field effect.

To enable the effect, see Enable Depth of Field (p. 2619).

### Contents
- Inputs (p. 2609)
- Outputs (p. 2609)
## Disable Depth Of Field

**Post Effects**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Disable Directional Blur

Disables the directional blur filter.

To enable the effect, see [Enable Directional Blur (p. 2621)](#).

### Contents

- Inputs (p. 2609)
- Outputs (p. 2609)
**Disable Effect Group**

Disables a specific effect group of settings that was enabled with the [Enable Effect Group](#) node. The **Group Name** must match the same name specified in the [Enable Effect Group](#) node.

For more information about effect group files, see [Customizing Post-Processing Effects](#).

![Disable Effect Group](image)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Group Name</td>
<td>String</td>
<td>Relative path to the effect group xml file.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the effect is disabled.</td>
</tr>
</tbody>
</table>

**Disable Frost**

Disables spots of frost on the screen.

To enable the effect, see [Enable Frost](#).

**Contents**

- Inputs (p. 2611)
- Outputs (p. 2611)
**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Disable Ghosting**

Disables ghosting effects.

To enable the effect, see Enable Ghosting (p. 2624).

**Contents**

- Inputs (p. 2611)
- Outputs (p. 2611)

---

**Disable Radial Blur**

Disables the radial blur effect.

To enable the effect, see Enable Radial Blur (p. 2626).

**Contents**

- Inputs (p. 2612)
- Outputs (p. 2612)
## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Disable Rain Drops

Disables the raindrop effects.

To enable the effect, see [Enable Rain Drops](p. 2628).

### Contents

- Inputs (p. 2612)
- Outputs (p. 2612)
Disable Sharpen

Disables the sharpen filter.

To enable the effect, see Enable Sharpen (p. 2629).

Contents

- Inputs (p. 2613)
- Outputs (p. 2613)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Disable Visual Artifacts

Disables the visual artifact effects.

To enable the effect, see Enable Visual Artifacts (p. 2630).

Contents

- Inputs (p. 2614)
- Outputs (p. 2614)
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Disable Volumetric Scattering

Disables volumetric effects.

To enable the effect, see Enable Volumetric Scattering (p. 2632).

Contents

- Inputs (p. 2614)
- Outputs (p. 2614)

Disable Water Droplets

Disables water droplet effects.

To enable the effect, see Enable Water Droplets (p. 2633).

Contents

- Inputs (p. 2615)
- Outputs (p. 2615)
**Disable Water Droplets**

Post Effects

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the effect is finished.</td>
</tr>
</tbody>
</table>

**Disable Water Flow**

Disables water flow effects.

To enable the effect, see *Enable Water Flow (p. 2634)*.

**Contents**

- Inputs (p. 2615)
- Outputs (p. 2615)
Enable Blur

Sets the blur filter, which uses Gaussian blur.

To disable the effect, see Disable Blur (p. 2607).

Contents

- Inputs (p. 2616)
- Outputs (p. 2617)

![Enable Blur Node](image)

Example

![Example Image](image)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
## Enable Color Correction

Sets color correction parameters. You can use this node to specify the CMYK, brightness, contrast, saturation, and hue in a scene. Most color correction properties aren't updated smoothly, so it's recommended that you hide stronger color correction changes with cuts or fading between scenes.

You can also apply color correction in the Track View. See Color Correction Node (p. 1599).

To disable the effect, see Disable Color Correction (p. 2608).

### Contents
- **Inputs** (p. 2618)
- **Outputs** (p. 2619)

### Example

In the following example, the **Saturation** value is 2.0.
## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Cyan</td>
<td>Number</td>
<td>Adjusts cyan to enhance the color of the scene.</td>
</tr>
<tr>
<td>Magenta</td>
<td>Number</td>
<td>Adjusts magenta to enhance color of the scene.</td>
</tr>
<tr>
<td>Yellow</td>
<td>Number</td>
<td>Adjusts yellow to enhance color of the scene.</td>
</tr>
<tr>
<td>Luminance</td>
<td>Number</td>
<td>Adjusts luminance to enhance the color of the scene.</td>
</tr>
<tr>
<td>Brightness</td>
<td>Number</td>
<td>Adjusts brightness to enhance light and darkness of the scene.</td>
</tr>
<tr>
<td>Contrast</td>
<td>Number</td>
<td>Adjusts contrast to enhance the bias of highlights and shadows of the scene.</td>
</tr>
<tr>
<td>Saturation</td>
<td>Number</td>
<td>Adjusts saturation to enhance the color intensity of the scene.</td>
</tr>
<tr>
<td>Hue</td>
<td>Number</td>
<td>Adjusts hue to enhance the color globally.</td>
</tr>
</tbody>
</table>
Enable Depth of Field

Configures the depth of field (DOF) effect, which gives you control over distance, range, and amount. You can use the node to add realism to scenes by simulating the way a real-world camera works. You can use a broad depth of field to focus on the entire scene, or use a shallow depth of field to have sharp focus only on objects that are a specific distance from the camera.

To disable the effect, see Disable Depth of Field (p. 2608).

You can also apply depth of field in the Track View. See Adding a Depth of Field Node (p. 1600).

Contents

- Inputs (p. 2620)
- Outputs (p. 2621)
Example

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Focus Distance</td>
<td>Number</td>
<td>Distance the focus is from the camera. Positive values are in front of the camera while negative values are behind the camera.</td>
</tr>
<tr>
<td>Focus Range</td>
<td>Number</td>
<td>Distance toward and away from the camera until maximum blurriness is reached.</td>
</tr>
<tr>
<td>Blur Amount</td>
<td>Number</td>
<td>Maximum blurriness value.</td>
</tr>
<tr>
<td>CoC Scale</td>
<td>Number</td>
<td>Sets the circle of confusion scale when a cone of light in a specific area appears unfocused or blurry. For example, a cone of light from a lens focuses on a specific point, but the light does not appear in perfect focus. This is also known as the blur circle of a blur spot.</td>
</tr>
<tr>
<td>Center Weight</td>
<td>Number</td>
<td>Sets the central samples weight.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Enable Directional Blur**

Applies a directional blur filter. To enable this node, you must enable the `r_MotionBlur` console variable. For more information, see Using the Console Window (p. 210).

To disable the effect, see Disable Directional Blur (p. 2609).

**Contents**

- Inputs (p. 2622)
- Outputs (p. 2622)
Example

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Direction</td>
<td>Vector2</td>
<td>Indicates the direction and strength of the blur, in the screen space.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Enable Effect Group

Enables a specific group of effects setting that are defined in an `xml` file.

**Note**

You must use the **Apply Effect Group At Position (p. 2606)** node for effect group `xml` files that include the `fadeDistance` parameter.

To disable the effect, see **Disable Effect Group (p. 2610)**.
For more information about effect group files, see Customizing Post-Processing Effects (p. 1769).

Contents
- Inputs (p. 2623)
- Outputs (p. 2623)

**Enable Effect Group**

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Group Name</td>
<td>String</td>
<td>Relative path to the effect group xml file.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the effect starts.</td>
</tr>
</tbody>
</table>

**Enable Frost**

Display spots of frost on the screen.

To disable the effect, see Disable Frost (p. 2610).

Contents
- Inputs (p. 2624)
- Outputs (p. 2624)
Example

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Amount of frost. Default value: 0</td>
</tr>
<tr>
<td>Center Amount</td>
<td>Number</td>
<td>Amount of frost at the center of the screen. Default value: 1</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Enable Ghosting

Apply a ghosting effect that overlaps and blurs previous frames together.

To disable the effect, see [Disable Ghosting (p. 2611)](#).
Contents
- Inputs (p. 2625)
- Outputs (p. 2626)

Example

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Strength of the ghosting effect.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Enable Radial Blur

Enables the radial blur filter around a defined 2D position on the screen. To enable this feature, you must enable the `r_MotionBlur` console variable. For more information, see Using the Console Window (p. 210).

To disable the effect, see Disable Radial Blur (p. 2611).

Contents

- Inputs (p. 2627)
- Outputs (p. 2627)
Example

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Amount of blurring.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Screen Position</td>
<td>Vector2</td>
<td>Screen position of the center of the blur.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: ((0.5, 0.5)) This is the center of the screen.</td>
</tr>
<tr>
<td>Radius</td>
<td>Number</td>
<td>Blurring radius.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Enable Rain Drops

Apply raindrops on the screen.

To disable the effect, see Disable Rain Drops (p. 2612).

Contents

- Inputs (p. 2629)
- Outputs (p. 2629)

Example
**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
| Amount       | Number | Enables raindrops. A value of 0 disables raindrops.  
|              |        | Default value: 1                                |
| Spaw Time Distance | Number | Sets the spawn time distance for the raindrops.  
|              |        | Default value: 0.35                             |
|              |        | For example, a value of 0.35 means the node waits at least 0.35 seconds between drops. |
| Size         | Number | Specifies the raindrop size.                    |
|              |        | Default value: 5.0                              |
| Size Variation| Number | Sets raindrops variation.                       |
|              |        | Default value: 2.5                              |

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Enable Sharpen**

Applies a sharpen filter. You can specify a negative value to blur the screen.

To disable the effect, see Disable Sharpen (p. 2613).

**Contents**

- Inputs (p. 2630)
- Outputs (p. 2630)
Example

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Amount of sharpening.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Enable Visual Artifacts

Applies effects associated with old television sets, such as grain, vsync, and interlacing. You can mask the effect with a texture or apply it to the entire screen.

To disable the effect, see Disable Visual Artifacts (p. 2613).

Contents

• Inputs (p. 2631)
• Outputs (p. 2632)

**Enable Visual Artifacts**

**Post Effects**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>VSync</td>
<td>Number</td>
<td>Amount of vsync (vertical sync) visible.</td>
</tr>
<tr>
<td>VSync Frequency</td>
<td>Number</td>
<td>Frequency of the vertical sync.</td>
</tr>
<tr>
<td>Interlacing</td>
<td>Number</td>
<td>Amount of interlacing visible.</td>
</tr>
<tr>
<td>Interlacing Tiling</td>
<td>Number</td>
<td>Amount of interlacing tiling.</td>
</tr>
<tr>
<td>Interlacing Rotation</td>
<td>Number</td>
<td>Amount of interlacing rotation.</td>
</tr>
<tr>
<td>Sync Wave Phase</td>
<td>Number</td>
<td>Sync wave phase.</td>
</tr>
<tr>
<td>Sync Wave Frequency</td>
<td>Number</td>
<td>Sync wave frequency.</td>
</tr>
<tr>
<td>Sync Wave Amplitude</td>
<td>Number</td>
<td>Sync wave amplitude.</td>
</tr>
<tr>
<td>Chroma Shift</td>
<td>Number</td>
<td>Chromatic shift.</td>
</tr>
</tbody>
</table>

**Inputs**

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>VSync</td>
<td>Number</td>
<td>Amount of vsync (vertical sync) visible.</td>
</tr>
<tr>
<td>VSync Frequency</td>
<td>Number</td>
<td>Frequency of the vertical sync.</td>
</tr>
<tr>
<td>Interlacing</td>
<td>Number</td>
<td>Amount of interlacing visible.</td>
</tr>
<tr>
<td>Interlacing Tiling</td>
<td>Number</td>
<td>Amount of interlacing tiling.</td>
</tr>
<tr>
<td>Interlacing Rotation</td>
<td>Number</td>
<td>Amount of interlacing rotation.</td>
</tr>
<tr>
<td>Sync Wave Phase</td>
<td>Number</td>
<td>Sync wave phase.</td>
</tr>
<tr>
<td>Sync Wave Frequency</td>
<td>Number</td>
<td>Sync wave frequency.</td>
</tr>
<tr>
<td>Sync Wave Amplitude</td>
<td>Number</td>
<td>Sync wave amplitude.</td>
</tr>
<tr>
<td>Chroma Shift</td>
<td>Number</td>
<td>Chromatic shift.</td>
</tr>
</tbody>
</table>
### Pin Type Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grain</td>
<td>Number</td>
<td>Amount of image grain.</td>
</tr>
<tr>
<td>Color Tint</td>
<td>Number</td>
<td>Amount of color tinting.</td>
</tr>
<tr>
<td>Texture Name</td>
<td>String</td>
<td>Visual artifacts mask texture.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

#### Enable Volumetric Scattering

Adds a volumetric effect for simulating fog, snow, or other environments. You can specify the color, speed, and amount for each effect, so that you can simulate various environments, such as lava.

**Note**

This effect has high performance impact and can negatively affect the frame rate.

To disable the effect, see [Disable Volumetric Scattering](p. 2614).

#### Contents

- Inputs (p. 2632)
- Outputs (p. 2633)

---

**Enable Volumetric Scattering**

*Post Effects*

- **In** [Amount: 1.0000]
- **Out**

---

#### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Sets volumetric scattering amount.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
</tbody>
</table>
### Pin | Type | Description
--- | --- | ---
Tiling | Number | Sets volumetric scattering tiling. Default value: 1
Speed | Number | Sets volumetric scattering animation speed. Default value: 1
Color | Color | Sets volumetric scattering color tint.

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Out | Event | Sends when the node is finished.

### Enable Water Droplets

Apply a water effect that appears from various sources on the screen. Unlike the [Enable Rain Drops (p. 2628)] node, this node simulates a splash of water thrown on the screen. For example, you can use this node when the camera leaves the water.

To disable the effect, see [Disable Water Droplets (p. 2614)].

### Contents

- Inputs (p. 2634)
- Outputs (p. 2634)
Example

![Example Image](image_url)

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Amount of water.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 5</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the effect starts.</td>
</tr>
</tbody>
</table>

**Enable Water Flow**

Draws water flowing down the entire screen. You can use this node in situations where the player is standing under flowing water, such as a waterfall.

To disable the effect, see Disable Water Flow (p. 2615).
• Inputs (p. 2635)
• Outputs (p. 2636)

Example

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Amount</td>
<td>Number</td>
<td>Amount of water.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Screen Fader**

Controls fading the screen to a color and/or a texture. You can use this node to script cinematic transitions, such as fading to black or fading to a loading screen texture. The fade is a screen overlay that is rendered on top of the scene. The size and position of this overlay can be adjusted to a different size than full screen. This is useful if you want to draw letter boxes at the top and bottom of the screen. If you use custom positioning with a texture, you can use this node as a quick way to draw decorative borders around the frame.

To use the **Screen Fader** node in the Track View, see Screen Fader Node (p. 1600).

**Topics**
- Inputs (p. 2637)
- Outputs (p. 2639)
- ScreenFader EBuses (p. 2639)

**Example**

In the following script, the screen first fades in, out, and then fades in again.
Example

See the following screen fading in a game.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fade Out</td>
<td>Event</td>
<td>Triggers fading out to a color or texture.</td>
</tr>
<tr>
<td>Fade In</td>
<td>Event</td>
<td>Triggers fading back in from a color or texture.</td>
</tr>
<tr>
<td>Fader Id</td>
<td>Number</td>
<td>Specifies which fader to use. This lets you maintain separate settings and/or layer fades on top of each other.</td>
</tr>
<tr>
<td>Pin</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Fade In Time</td>
<td>Number</td>
<td>Number of seconds when fading in.</td>
</tr>
<tr>
<td>Fade Out Time</td>
<td>Number</td>
<td>Number of seconds when fading out.</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>The color to fade to and from. The alpha channel is ignored. The use of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this property depends on the value of <strong>Use Current Color</strong>, and whether</td>
</tr>
<tr>
<td></td>
<td></td>
<td>you are fading in or fading out.</td>
</tr>
<tr>
<td>Use Current Color</td>
<td>Boolean</td>
<td>Specify true to continue to use the current color for the fade overlay.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The use of the <strong>Color</strong> property depends on the value of **Use Current</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color**, and whether you are fading in or fading out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Current Color</strong> is the color that is rendered by the fader, including the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>alpha channel. This is generally whatever color is left over from prior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fading activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The <strong>Color Property Behavior</strong> (p. 2638) table in the next section shows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the color values that are blended to create the fading transition.</td>
</tr>
<tr>
<td>Texture Name</td>
<td>String</td>
<td>(Optional) The name of a texture from or to which to fade. To use only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>solid colors for fading, leave this value empty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information, see <strong>Finding the Texture Name</strong> (p. 2682).</td>
</tr>
<tr>
<td>Update Always</td>
<td>Boolean</td>
<td>Continues fading even when the game is paused.</td>
</tr>
<tr>
<td>Screen Coordinates</td>
<td>Vector4</td>
<td>Sets the screen coordinates (left, top, right, and bottom) where the fade</td>
</tr>
<tr>
<td></td>
<td></td>
<td>overlay is drawn.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default is full screen (0, 0, 1, 1).</td>
</tr>
</tbody>
</table>

**Color Property Behavior**

The following table shows the actual color values that are blended to cause the fading transition.

- **Fading** – Refers to whether a fade-in or fade-out effect was triggered.
- **Use Current Color** – The node's **Use Current Color** input setting.
- **Start Color** – The color used to render the overlay at the beginning of the transition.
- **Final Color** – The color used at the end of the transition.
- **Color** – The node's **Color** input setting.
- **Current Color** – The color that the fader actively renders, including the alpha channel. This is generally whatever color is left over from prior fading activity.

<table>
<thead>
<tr>
<th>Fading</th>
<th>Use Current Color Input</th>
<th>Start Color</th>
<th>Final Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>true</td>
<td><strong>Current Color</strong></td>
<td><strong>Current Color</strong> with Alpha=0</td>
</tr>
<tr>
<td>In</td>
<td>false</td>
<td><strong>Color</strong> input with Alpha=1</td>
<td><strong>Color</strong> input with Alpha=0</td>
</tr>
<tr>
<td>Out</td>
<td>true</td>
<td><strong>Current Color</strong></td>
<td><strong>Current Color</strong> with Alpha=1</td>
</tr>
</tbody>
</table>
Fading

<table>
<thead>
<tr>
<th>Input</th>
<th>Start Color</th>
<th>Final Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>false</td>
<td>Color input with Alpha=0</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fade Out Complete</td>
<td>Event</td>
<td>Sent when fade out is complete.</td>
</tr>
<tr>
<td>Fade In Complete</td>
<td>Event</td>
<td>Sent when fade in is complete.</td>
</tr>
<tr>
<td>Current Color</td>
<td>Color</td>
<td>Outputs the current value for the fade overlay color.</td>
</tr>
</tbody>
</table>

**Note**

Most Script Canvas graphics features are available in Lua. However, ScreenFader is a single node in Script Canvas, and an EBus in Lua. For more information, see the ScreenFader EBuses (p. 2639).

**ScreenFader EBuses**

Provides the Lua scripting equivalent to the Screen Fader (p. 2636) node in Script Canvas.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**Contents**

- ScreenFaderRequestBus FadeOut (p. 2639)
- ScreenFaderRequestBus FadeIn (p. 2640)
- ScreenFaderRequestBus SetTexture (p. 2640)
- ScreenFaderRequestBus SetScreenCoordinates (p. 2640)
- ScreenFaderRequestBus GetCurrentColor (p. 2640)
- ScreenFaderNotificationBus OnFadeOutComplete (p. 2641)
- ScreenFaderNotificationBus OnFadeInComplete (p. 2641)
- ScreenFaderManagementRequestBus GetNumFaderIDs (p. 2641)

**ScreenFaderRequestBus FadeOut**

Triggers fading out to a solid color.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Color</td>
<td>The color to fade out to. The use of the Color property depends on the value of Use Current Color, and whether you are fading in or fading out.</td>
</tr>
<tr>
<td>duration</td>
<td>Float</td>
<td>Duration of the fade out in seconds.</td>
</tr>
<tr>
<td>useCurrentColor</td>
<td>Boolean</td>
<td>When true, the transition uses the current color for the fade overlay. When false, the transition begins fully transparent.</td>
</tr>
</tbody>
</table>
Parameter | Type     | Description
---         | ---      | ---
Current Color | is the color that is rendered by the fader, including the alpha channel. This is generally whatever color is left over from prior fading activity.
The Color Property Behavior (p. 2638) table shows the color values that are blended to create the fading transition.

updateAlways | Boolean | Continues fading even when the game is paused.

ScreenFaderRequestBus FadeIn
Triggers fading in to a solid color.

Parameter | Type     | Description
---         | ---      | ---
color | Color | The color to fade through. Ignored if Use Current Color is true.
duration | Float | Duration of the fade in seconds.
useCurrentColor | Boolean | When true, the transition uses the current color for the fade overlay. When false, the transition begins from the target color.
updateAlways | Boolean | Continues fading even when the game is paused.

ScreenFaderRequestBus SetTexture
Sets a texture for the fade overlay.

Parameter | Type     | Description
---         | ---      | ---
textureName | String | The name of a texture from or to which to fade. To clear the texture, specify an empty string.
For more information, see Finding the Texture Name (p. 2682).

ScreenFaderRequestBus SetScreenCoordinates
Sets the screen coordinates where the fade overlay is drawn.

Parameter | Type     | Description
---         | ---      | ---
screenCoordinates | Vector4 | Sets the screen coordinates (left, top, right, and bottom) where the fade overlay is drawn.
The default is full screen (0, 0, 1, 1).

ScreenFaderRequestBus GetCurrentColor
Returns the current color of the fade overlay.
**ScreenFaderNotificationBus OnFadeOutComplete**

A callback function that is called when fading out is complete.

**ScreenFaderNotificationBus OnFadeInComplete**

A callback function that is called when fading in is complete.

**ScreenFaderManagementRequestBus GetNumFaderIDs**

Returns the number of available fader IDs. The number returned is not necessarily the number of faders that have been created.

**Set Color Chart**

Applies a color chart texture for color grading.

**Note**

When you call the node for the first time, you must set a default color chart for the fade to work correctly. The Set Color Chart node uses this color chart as a reference to fade into the next color chart. If the Set Color Chart node doesn't have a default color chart to fade from, the node will immediately fade to the first color chart, regardless of the fade time.

For an example script, see Example Set Color Chart Script (p. 2646).

**Contents**

- Inputs (p. 2641)
- Creating a Color Chart (p. 2642)
- Console Variables for Color Charts (p. 2647)

---

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Texture Name</td>
<td>String</td>
<td>The name of a color chart texture. For more information, see Finding the Texture Name (p. 2682).</td>
</tr>
<tr>
<td>Fade Time</td>
<td>Number</td>
<td>Number of seconds to fade into the color grading.</td>
</tr>
</tbody>
</table>

---

Version 1.28

2641
Creating a Color Chart

You can create a color chart to apply color grading to your project. A color chart uses a reference image that can be an example image from your game or an image that contains a wide variety of color. You then modify the image, such as changing the hue, saturation, brightness, and so on.

When you specify the color chart file with the Set Color Chart (p. 2641) node, Lumberyard takes the image and its color chart and applies the color changes to your level. For example, you can use a color chart with negative saturation to make your game appear dark and muted.

To create a color chart

1. Make a copy of the color chart file default_cch.tiff. You can find the color chart file in the lumberyard_version\dev\Engine\EngineAssets\Textures directory.

   Note
   You must use this exact file so that the Resource Compiler can detect the color chart and use it to process the color charts that you create.

   The following is an example color chart image:

2. Use a reference image that shows a wide range of colors and that has not been color corrected yet, and do the following:

   Note
   • You don't need to use a high resolution or large reference image.
   • You can use any image as a reference, such as screenshot from your game.

   a. Copy and paste the color chart image into the reference image.
   b. Flatten all layers for the image.
   c. Save the image as filename_cch.tif, such as default_startergame_cch.tif.

      Note
      You do not need to include the _cch suffix for the image file.

Example default color chart image

The following image is from Starter Game Sample (p. 148) and includes the color chart in the bottom right.
3. Make a copy of the image and then do the following:

   a. Modify the image so that it has the color correction that you want. Avoid extreme color correction changes, which can result in overlapping colors when the scene is processed.

   b. Save the image as `filename_cch.tif`, such as `saturation_cch.tif`.

**Example modified color chart image**

The following image lowers the saturation.

4. Move all three files (`default_cch.tif` and your color charts) to your game project directory, such as `lumberyard_version\dev\StarterGame\textures\defaults\`. 
4. You use this directory to specify the file path to the color chart file in the **Set Color Chart** node. For more information, see **Finding the Texture Name** (p. 2682).

5. To compile your images so that Lumberyard Editor recognizes them as color charts, do one of the following:

   - In Lumberyard Editor, in the **Asset Browser**, navigate to the color chart file and double-click it.
   - If you have the RC Shell Commands plugin installed, right-click the image and choose **RC Compile Image**.

6. In the Resource Compiler, verify that **ColorChart** is selected, click **Generate Output**, and then click **OK**.

**Note**
You can install the plugin in Lumberyard Setup Assistant. For more information, see **Using Lumberyard Setup Assistant to Set Up Your Development Environment** (p. 16).
Note

- You can clear the **Tiled** option to preview the image correctly. This option does not affect your color chart.

- To see more information about **ColorChart** settings, click **Show preset info**. You can find these settings in the rc.ini file, located in the `lumberyard_version\dev\Bin64\vc141\rc` directory.
For more information, see Using the Resource Compiler Image Tool (p. 283).

7. Repeat steps 5 and 6 for all your color chart images.

8. In the Script Canvas editor, create a script. In the Set Color Chart (p. 2641) nodes, specify the path to the color chart files that you created, and the Fade Time.

**Example Set Color Chart Script**

The example script does the following:

1. When the graph starts, the Set Color Chart node sets a default color chart with the `default_startergame_cch.tif` file.
2. The Delay node waits three seconds.
3. During the next five seconds, the Set Color Chart node fades the screen to the color chart with the `saturation_cch.tif` file.

9. Attach the Script Canvas (p. 818) component to an entity and specify the script. For more information, see Working with Components (p. 479).

10. To test your script in game mode, press Ctrl+G.

**Example**

The following example demonstrates the script.
11. To exit game mode, press Esc.

**Console Variables for Color Charts**

You can use the following console variables for working with color charts. For more information, see *Using the Console Window (p. 210).*

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_ColorGrading</td>
<td>Enables color grading.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>r_ColorGradingCharts</td>
<td>Enables color grading with color charts.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>r_ColorGradingChartsCache</td>
<td>Enables cache updates for the color grading charts.</td>
<td>0 = Always update.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Update every other frame.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Update every two frames.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = Update every three frames.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = Update every four frames (default).</td>
</tr>
<tr>
<td>r_ColorGradingFilters</td>
<td>Enables color grading for filters.</td>
<td>0 = Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = On (default)</td>
</tr>
<tr>
<td>r_ColorGradingLevels</td>
<td>Enables color grading for levels.</td>
<td>0 = Off</td>
</tr>
</tbody>
</table>
### Console Variable

<table>
<thead>
<tr>
<th>Console Variable</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_ColorGradingSelectiveColor</td>
<td>Enables color grading for the selected color.</td>
<td>0 = Off, 1 = On (default)</td>
</tr>
</tbody>
</table>

### Procedural Material Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard supports Allegorithmic Substance for working with procedurally-generated materials. Procedural materials are materials in which you can modify their textures during runtime.

You can use the Allegorithmic Substance Designer to author procedural materials and then import them into Lumberyard Editor. You can then use the Script Canvas editor or Lua to modify the procedural material's textures at runtime.

To enable this feature, you must enable the Allegorithmic Substance gem. For more information, see Add modular features and assets with Gems (p. 1064).

For more information, see Working with Substances (p. 1760) and Lumberyard Substance Integration.

To work with procedural materials in scripts

1. **Reference the procedural material file**
   - To reference the procedural material, use the Find by Name (p. 2650) or Find by Name and Index (p. 2652) node.

2. **Modify the parameter values for the procedural material**
   - Use the Set Input <Type> nodes to modify the parameter values for the procedural material. The values that you specify in the nodes modify the parameters that update the procedural material's textures. You can modify parameters for multiple procedural materials in a single frame.

   **Note**

   *Input parameters* are the parameters that you modify for the procedural material. You can find the parameters for a material in the Substance Editor. For example, the sample procedural material brickWall_04 has input parameters such as *Age, Mortar, and Depth*. To modify these input parameters during runtime, specify the parameter name and its value in a node such as Set Input Number (p. 2664).

3. **Render the procedural materials**
   - After you specify the changes that you want, use the Render Asynchronous (p. 2661) or Render Synchronous (p. 2662) node to update the procedural material's texture. The render node applies to all procedural materials in which their parameters have changed, so at most, call a render node once per frame.

**Example**

The Allegorithmic Substance gem includes a sample procedural material named brickWall_04. You can import this file into Lumberyard and write a script that dynamically changes the *Age* and *Mortar* input parameter values, so that the wall appears to erode over time.
See the following example script to enable this effect.

1. The **Find by Name and Index** node (1) retrieves the brickWall_04 procedural material and sends the material to all **Set Input Number** nodes (3, 4, 6, 7).
2. The **Duration** node (2) runs the aging process for four seconds through the **Out** pin, which is triggered every frame. The node finalizes the aging process through the **Done** pin.
3. The first **Set Input Number** node (3) transitions the **Age** parameter from 0 to 0.5 over four seconds.
4. The second **Set Input Number** node (4) transitions the **Mortar** parameter from 0 to 1 over four seconds.
5. The **Render Asynchronous** node (5) applies these changes to the procedural material's textures.

6. For the **Duration** node (2), the **Done** pin triggers one last update with the final **Age** value of 0.5 (node 6) and the **Mortar** value of 1.0 (node 7).

7. With the **Force** parameter enabled, the **Render Asynchronous** node (8) applies these changes to the procedural material's textures.

   **Note**

   Steps 6 and 7 are not always required, but are examples of how you can use the **Render Asynchronous** node's **Force** parameter to guarantee that a final update is applied. For more information, see **Render Asynchronous** (p. 2661).

**Topics**

- Find by Name (p. 2650)
- Find by Name and Index (p. 2652)
- Get Input Color (p. 2654)
- Get Input Number (p. 2655)
- Get Input String (p. 2656)
- Get Input Vector2 (p. 2657)
- Get Input Vector3 (p. 2658)
- Get Input Vector4 (p. 2658)
- On Render Finished (p. 2659)
- Render Asynchronous (p. 2661)
- Render Synchronous (p. 2662)
- Set Input Color (p. 2663)
- Set Input Number (p. 2664)
- Set Input String (p. 2665)
- Set Input Vector2 (p. 2666)
- Set Input Vector3 (p. 2667)
- Set Input Vector4 (p. 2668)

**Find by Name**

Finds a procedural material by the substance file name and graph name. All substance files contain one or more graphs, and these graphs correspond to a procedural material.

**Contents**

- Inputs (p. 2652)
- Outputs (p. 2652)
To find the substance name for a procedural material

1. In Lumberyard Editor, choose **Tools, Plug-ins, Substance Editor**.
2. Browse to the substance material that you want. The substance material name appears in the **Substance** field and the graph name appears in the tab.

**Example**

The substance material path is `materials/substance/brickwall_04.sbsar` and the graph name is `BrickWall_04`.

3. Copy the substance material path and in the **Find by Name** node, do the following:
   a. For **Substance Name**, paste the substance material path.
   b. For **Graph Name**, enter the graph name.
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Substance Name</td>
<td>String</td>
<td>The path name of the substance material .sbsar or .smtl file to find.</td>
</tr>
<tr>
<td>Graph Name</td>
<td>String</td>
<td>Name of the specific graph in the substance material.</td>
</tr>
<tr>
<td>Force Load</td>
<td>Boolean</td>
<td>Forces the procedural material to load, if it's not already loaded.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The specified procedural material. Returns Invalid if the material can't be found.</td>
</tr>
</tbody>
</table>

Find by Name and Index

Finds a procedural material by the substance file name and the index of a graph. All substance files contain one or more graphs, and these graphs correspond to a procedural material.

Contents

- Inputs (p. 2653)
- Outputs (p. 2654)
To find the substance name

1. In Lumberyard Editor, choose **Tools, Plug-ins, Substance Editor**.
2. Browse to the substance material that you want. The substance material name appears in the **Substance** field and the graph name appears in the tab.

**Example**

The substance material path is `materials/substance/brickwall_04.sbsar` and the graph name `BrickWall_04` appears in the tab.

3. Copy the substance material path and in the **Find by Name and Index** node, do the following:
   a. For **Substance Name**, paste the substance material path.
   b. For **Graph Index**, specify the graph that you want. The far-left tab is index 0, the next is 1, and so on. If the substance material has only one graph, leave the **Graph Index** field as 0.

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Substance Name</td>
<td>String</td>
<td>The path name of the substance material <code>.sbsar</code> or <code>.smtl</code> file to find.</td>
</tr>
<tr>
<td>Graph Index</td>
<td>Number</td>
<td>Index of the specific graph in the substance material. Default value: 0</td>
</tr>
<tr>
<td>Force Load</td>
<td>Boolean</td>
<td>Forces the procedural material to load, if it's not already loaded.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The specified procedural material to return.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns Invalid if the material can't be found.</td>
</tr>
</tbody>
</table>

Get Input Color

Returns the value of an input parameter as a color. The color channels are assumed to be a floating point, including the alpha channel. Use this node with the Float4 substance data type.

**Note**
You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are assigned a value of 0.

The node does not perform any range conversion. For example, if an Integer4 substance data type has a value of (255, 0, 0, 255), this is returned as (255.0, 0.0, 0.0, 255.0), which is outside the normal 0.0 to 1.0 range of a color value.

Contents
- Inputs (p. 2654)
- Outputs (p. 2655)
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Color</td>
<td>Color</td>
<td>The value of the specified input parameter. Returns (0.0, 0.0, 0.0, 0.0) if the value can't be found.</td>
</tr>
</tbody>
</table>

Get Input Number

Returns the value of an input parameter as a number. Use this node with the Float1 and Integer1 substance data types.

**Note**
You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the dimensions don't match, the extra dimensions from the input parameter are ignored.

Contents
- Inputs (p. 2655)
- Outputs (p. 2656)

![Get Input Number Node](image)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to query.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to return.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Number</td>
<td>Number</td>
<td>The value of the specified input parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns 0 if the value can't be found.</td>
</tr>
</tbody>
</table>

Get Input String

Returns the value of an input parameter as a string or image.

If the input parameter is a string type, the node returns the text value. If the input parameter is an image type, the node returns the path name of the image file.

Contents
- Inputs (p. 2656)
- Outputs (p. 2656)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to query.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to return.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
## Get Input Vector2

Returns the value of an input parameter as a Vector2. Use this node with the Float2 and Integer2 substance data types.

**Note**
You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are assigned a value of 0. If there are more dimensions, they are ignored.

### Contents
- Inputs (p. 2657)
- Outputs (p. 2657)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to query.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to return.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector2</td>
<td>Vector2</td>
<td>The value of the specified input parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns 0 if the value can't be found.</td>
</tr>
</tbody>
</table>
Get Input Vector3

Returns the value of an input parameter as a Vector3. Use this node with the Float3 and Integer3 substance data types.

**Note**
You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are assigned a value of 0. If there are more dimensions, they are ignored.

**Contents**
- Inputs (p. 2658)
- Outputs (p. 2658)

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to query.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to return.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector3</td>
<td>Vector3</td>
<td>The value of the specified input parameter. Returns 0 if the value can't be found.</td>
</tr>
</tbody>
</table>

Get Input Vector4

Returns the value of an input parameter as a Vector4. Use this node with the Float4 and Integer4 substance data types.
Note
You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are assigned a value of 0.

Contents
- Inputs (p. 2659)
- Outputs (p. 2659)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to query.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to return.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Vector4</td>
<td>Vector4</td>
<td>The value of the specified input parameter.  Returns 0 if the value can't be found.</td>
</tr>
</tbody>
</table>

**On Render Finished**

This node is triggered when an update for a procedural material finishes rendering.

You can use the node to detect when texture updates scheduled by the Render Asynchronous (p. 2661) node are complete.
Contents

- Outputs (p. 2661)

To verify that the render update is the event that you want, compare the Render ID values that the Render Asynchronous (p. 2661) and On Render Finished nodes return. You can use the Equal To (==) node, such as in the following example.
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when a procedural material render update is complete.</td>
</tr>
<tr>
<td>Render ID</td>
<td>Number</td>
<td>Unique ID for the scheduled render process.</td>
</tr>
</tbody>
</table>

**Render Asynchronous**

Schedules all procedural materials that have pending parameter changes to regenerate their texture maps. This update is done asynchronously, so that the textures are updated within a few frames. Use this node (instead of the **Render Synchronous** (p. 2662) node) for changes made during gameplay.

The node returns a **Render ID**, which you can use with the **On Render Finished** (p. 2659) node to detect when the textures finish updating. This should only be necessary if you need to synchronize another action with the texture update.

If the **Force** parameter is disabled, the node renders only procedural materials that are not currently rendering. Use this option when updating procedural materials every frame. Otherwise, the render queue can back up and may not catch up.

When the **Force** parameter is enabled, the node is guaranteed to render the changes. Use this option to make only occasional updates to the procedural material, or if you previously updated the procedural material every frame and you want a final update to guarantee that the update is not missed.

For an example script with the **Render Asynchronous** node, see the example script (p. 2649).

**Note**

If the **Set Input <Type>** nodes don't modify a procedural material, which means the procedural material doesn't have pending parameter changes, the **Render Asynchronous** node ignores the material.

**Contents**

- **Inputs** (p. 2661)
- **Outputs** (p. 2662)

---

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
### Pin Types

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Force</td>
<td>Boolean</td>
<td>Forces the render to be scheduled. Specify <code>false</code> for better performance on high frequency updates.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Render ID</td>
<td>Number</td>
<td>Unique ID for the scheduled render process. The value is 0 if a render was not scheduled.</td>
</tr>
</tbody>
</table>

### Render Synchronous

Regenerates the texture maps for all procedural materials that have pending parameter changes. This update is done synchronously, so that the textures are updated by the time the node completes. In general, use this node in cases where the frame rate isn't a concern and you want to avoid the complexity of the [Render Asynchronous](p. 2661) node.

**Note**

If the [Set Input <Type>](p. 2662) nodes don't modify a procedural material, which means the procedural material doesn't have pending parameter changes, the [Render Synchronous](p. 2661) node ignores the material.

### Contents

- [Inputs](p. 2662)
- [Outputs](p. 2663)

### Render Synchronous

- **Procedural Material**
  - In
  - Out

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Set Input Color**

Sets the value of an input parameter as a color. The color channels are assumed to be floating point, including an alpha channel. Use this node with the `Float4` substance data type.

**Note**

You can use this node with all substance data types: `Float1` through `Float4` and `Integer1` through `Integer4`. However, if the input parameter has fewer dimensions, the extra dimensions are ignored.

If the input parameter is an integer type, the numbers that you specify are rounded down to the nearest integer value (for example, a value of 4.7 resolves to 4). For example, if you use a 50% gray opaque color to set a `Float4`, this results in a value of (0.5, 0.5, 0.5, 1.0). If you use the same color to set an `Integer4`, this results in a value of (0, 0, 0, 1).

**Contents**

- Inputs (p. 2663)
- Outputs (p. 2664)
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Set Input Number**

Sets the value of an input parameter as a number. You can use this node with the Float1 and Integer1 substance data types.

If the input parameter is an integer type, the numbers that you specify are rounded down to the nearest integer value. For example, a value of 4.7 resolves to 4.

**Note**

You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has more dimensions, the extra dimensions are assigned a value of 0.

**Contents**

- Inputs (p. 2664)
- Outputs (p. 2665)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to modify.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to set.</td>
</tr>
<tr>
<td>Number</td>
<td>Number</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

## Set Input String

Sets the value for an input parameter as a string or an image.

If the input parameter is a string type, the node sets the text value that the procedural material uses.

If the input parameter is an image type, this sets the path name of the image file that the procedural material uses.

### Contents
- Inputs (p. 2665)
- Outputs (p. 2665)

## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to modify.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to set.</td>
</tr>
<tr>
<td>String</td>
<td>String</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Set Input Vector2

Sets the value of an input parameter as a Vector2. Use this node with the Float2 and Integer2 substance data types.

If the input parameter is an integer type, the numbers that you specify are rounded down to the nearest integer value. For example, a value of 4.7 resolves to 4.

**Note**

You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are ignored. If the input parameter has more dimensions, they are assigned a value of 0.

**Contents**

- Inputs (p. 2666)
- Outputs (p. 2666)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to modify.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to set.</td>
</tr>
<tr>
<td>Vector2</td>
<td>Vector2</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Set Input Vector3

Sets the value of an input parameter as a Vector3. Use this node with the Float3 and Integer3 substance data types.

If the input parameter is an integer type, the numbers that you specify are rounded down to the nearest integer value. For example, a value of 4.7 resolves to 4.

**Note**

You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are ignored. If the input parameter has more dimensions, they are assigned a value of 0.

**Contents**

- Inputs (p. 2667)
- Outputs (p. 2667)

![Set Input Vector3](image)

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to modify.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to set.</td>
</tr>
<tr>
<td>Vector3</td>
<td>Vector3</td>
<td>The new value to apply.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Set Input Vector4

Sets the value of an input parameter as a Vector4. Use this node with the Float4 and Integer4 substance data types.

If the input parameter is an integer type, the numbers that you specify are rounded down to the nearest integer value. For example, a value of 4.7 resolves to 4.

**Note**
You can use this node with all substance data types: Float1 through Float4 and Integer1 through Integer4. However, if the input parameter has fewer dimensions, the extra dimensions are ignored.

**Contents**
- Inputs (p. 2668)
- Outputs (p. 2668)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Procedural Material</td>
<td>Procedural material</td>
<td>The procedural material to modify.</td>
</tr>
<tr>
<td>Input Name</td>
<td>String</td>
<td>The name of the input parameter to set.</td>
</tr>
<tr>
<td>Vector4</td>
<td>Vector4</td>
<td>The new value to specify.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>
Shadow Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the following shadow nodes in the Script Canvas editor to configure shadow effects.

Topics
- High Quality Shadow (p. 2669)
- Recompute Static Shadows (p. 2671)

High Quality Shadow

You can use the following High Quality Shadow nodes in the Script Canvas editor to configure effects with the High Quality Shadow (p. 614) component.

Topics
- Get Enabled (p. 2669)
- Set Enabled (p. 2670)

Get Enabled

Returns whether an entity's High Quality Shadow (p. 614) component is enabled.

Contents
- Inputs (p. 2669)
- Outputs (p. 2670)

Get Enabled

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
### Pin Type Description

| Source | EntityID | References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514). |

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Event</td>
<td>Returns true if the Script Canvas (p. 818) component is enabled.</td>
</tr>
</tbody>
</table>

### Set Enabled

Indicates whether the entity's High Quality Shadow (p. 614) component is enabled.

### Contents
- Inputs (p. 2670)
- Outputs (p. 2671)

### Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Source</td>
<td>EntityID</td>
<td>References a specific entity from which events are generated. By default, it references Self, the entity to which the Script Canvas (p. 818) component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Pin</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Canvas (p. 818)</strong> component attaches the current script. You can also select another entity. For more information, see Setting Entity Targets (p. 2514).</td>
</tr>
<tr>
<td>Enabled</td>
<td>Boolean</td>
<td>Indicates whether the <strong>Script Canvas (p. 818)</strong> component is enabled.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

**Recompute Static Shadows**

Triggers recalculation of cached shadow maps.

**Contents**

- Inputs (p. 2671)
- Outputs (p. 2672)

![Recompute Static Shadows](image)

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Bounds</td>
<td>AABB</td>
<td>Specifies the area where shadow maps should be recomputed.</td>
</tr>
<tr>
<td>Next Cascade Scale</td>
<td>Number</td>
<td>Multiplier for scaling the bounding boxes for subsequent cached cascades. The bounding boxes are scaled versions of the preceding cascades.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Time of Day Nodes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use time of day nodes in the **Script Canvas** editor to set the current time, change the speed of time progression, and load predefined time of day configuration files.

For more information about configuring time of day effects, see Adding Sky Effects (p. 1265) and Creating Time of Day Sky Effects (p. 1269).

**Topics**

- Get Speed (p. 2672)
- Get Time (p. 2673)
- Load Definition File (p. 2673)
- Set Speed (p. 2674)
- Set Time (p. 2675)

**Get Speed**

Returns the current time of day speed multiplier. The value is specified in conversion units of hours per second. For example, a value of 1 runs through an entire time of day cycle in 24 seconds.

**Contents**

- Inputs (p. 2672)
- Outputs (p. 2673)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Speed</td>
<td>Number</td>
<td>Current speed multiplier that determines how quickly the time of day changes.</td>
</tr>
</tbody>
</table>

Get Time

Returns the current time of day for the level. Time values are specified as decimal numbers from 0.0 to 24.0. For example, a value of 13.75 is 1:45 PM. A value of 0.0 or 24.0 is 12:00 AM.

Contents

- Inputs (p. 2673)
- Outputs (p. 2673)

Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Time</td>
<td>Number</td>
<td>Time of day in hours. Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0.0 to 24.0</td>
</tr>
</tbody>
</table>

Load Definition File

Loads a time of day preset XML file file_name.xml and then updates the sky. For example, the preset file can be named Time_Of_Day.xml.
Contents

- Inputs (p. 2674)
- Outputs (p. 2674)

## Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node to load the time of day preset .xml file.</td>
</tr>
<tr>
<td>File Name</td>
<td>String</td>
<td>Path to the time of day preset .xml file.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Returns true if the file loads successfully.</td>
</tr>
</tbody>
</table>

### Set Speed

Sets the current time of day speed multiplier. The value is specified in conversion units of hours per second.

For example, a value of 0.0 stops automatic time progression, although you can change the time manually with the **Set Time (p. 2675)** node.

A value of 1.0 runs through an entire time of day cycle in 24 seconds.

**Note**

- To make simulations similar to real-world time, specify a value of 0.0003.

Contents

- Inputs (p. 2675)
- Outputs (p. 2675)
### Set Speed

#### Time of Day

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Speed</td>
<td>Number</td>
<td>The speed multiplier that determines how quickly the time of day changes. Default value: 0. Valid values: 0.0 to 100.0</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Set Time

Sets a time value for the time of day system. Time values are specified as decimal numbers from 0.0 to 24.0. For example, a value of 13.75 is 1:45 P.M. A value of 0.0 or 24.0 is 12:00 A.M. Once the time of day is set, the time of day continues as specified by the speed setting.

#### Contents
- Inputs (p. 2676)
- Outputs (p. 2676)
Inputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Time</td>
<td>Number</td>
<td>Time of day in hours.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0.0 to 24.0</td>
</tr>
<tr>
<td>Force Update</td>
<td>Boolean</td>
<td>If true, the entire sky updates immediately in the current frame; otherwise, the sky is rendered across several frames.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: False</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

Material Variables

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you use a node such as Find by Name (p. 2580), Load by Name (p. 2586), or Clone (p. 2579), the script can only use the material for that one frame. After the frame, the material is then invalid. If you want to reuse the material in your script, you can store the material in a Variable Material node. You can also use this node when you want to find or load a material only once when the game starts.

For more information about creating variable nodes, see Managing Script Canvas Variables (p. 2482).

Note

To use a material variable node, you must assign the variable to a valid material with a node such as Find by Name (p. 2580), Load by Name (p. 2586), or Clone (p. 2579).

Contents

- Get Material Variable Node (p. 2677)
  - Inputs (p. 2677)
  - Outputs (p. 2677)
- Set Material Variable Node (p. 2677)
  - Inputs (p. 2677)
  - Outputs (p. 2678)
- Material Variable Node Example (p. 2678)
Get Material Variable Node

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>Sends the variable's current material.</td>
</tr>
</tbody>
</table>

Set Material Variable Node

**Inputs**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Event</td>
<td>Triggers the node.</td>
</tr>
<tr>
<td>Material</td>
<td>Material</td>
<td>Sets the variable's current material.</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Pin</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Event</td>
<td>Sends when the node is finished.</td>
</tr>
</tbody>
</table>

### Material Variable Node Example

The following example uses a material variable node to make a material flash red.

1. The **Variable Manager** defines a variable material node named *MyMaterial*.
2. The **Load by Name (p. 2586)** node specifies the `materials/gettingstarted/materials/grid` material file and the result of the node is stored in a variable with a **Set MyMaterial** node.
3. Every 0.5 seconds, the **Get MyMaterial** node uses the variable with the **Set Param Color (p. 2587)** node to change the material's **diffuse** parameter from red to white.

### Example

One entity has the `primitive_cylinder.cfg` mesh and the other entity has the `primitive_sphere.cfg` mesh. Both entities share the same `grid.mtl` material file. When the Script Canvas graph runs, it changes the appearance of every entity that has this material applied.
Finding the Material Name

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can specify material names to reference for nodes in the Script Canvas editor. The material name is generally the path that starts at the top asset directory but without the `.mtl` file name extension.

**To find the material name**

1. In Lumberyard Editor, choose Tools, Material Editor.
2. In the Material Editor, navigate to the material file.
3. Right-click the material file and choose Copy Path to Clipboard. This copies the path to the file.

**Example**

In the Samples Project, the material name for `logo_white.mtl` is `materials/gettingstartedmaterials/logo_white`. 
Finding the Material Parameter Name

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can specify material parameter names for some Script Canvas editor nodes. In the Material Editor, you can find the material parameter names in the following sections:

- **Opacity Settings**
- **Lighting Settings**
- **Shader Params**

You can find common parameters in the **Opacity Settings** and **Lighting Settings** sections. You can find parameters that are specific to the selected shader in the **Shader Params** section.
To find the material parameter name

1. In Lumberyard Editor, choose Tools, Material Editor.
2. In the Material Editor, navigate to the material file.
3. In the Opacity Settings, Lighting Settings, and Shader Params sections, pause on a parameter to find the script parameter name.

Example

For Diffuse Color (Tint), the script parameter name is diffuse.

Example

For Emittance Map Gamma, the script parameter name is EmittanceMapGamma.
The following parameter names are commonly available and built in all materials.

### Material Param Names

<table>
<thead>
<tr>
<th>Param</th>
<th>Type</th>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opacity</td>
<td>Float</td>
<td>Opacity</td>
<td>Sets the transparency amount. Uses 0 to 99 to set Alpha Blend and 100 for Opaque and Alpha Test.</td>
</tr>
<tr>
<td>alpha</td>
<td>Float</td>
<td>AlphaTest</td>
<td>Uses the alpha mask and refines the transparent edge. Uses 0 to 50 to bias toward white or 50 to 100 to bias toward black.</td>
</tr>
<tr>
<td>diffuse</td>
<td>Color</td>
<td>Diffuse Color (Tint)</td>
<td>Tints the material diffuse color. Physically based materials should be left at white.</td>
</tr>
<tr>
<td>specular</td>
<td>Color</td>
<td>Specular Color</td>
<td>Shininess and color of reflective highlights.</td>
</tr>
<tr>
<td>shininess</td>
<td>Float</td>
<td>Smoothness</td>
<td>Smoothness or glossiness simulating how light bounces off the surface.</td>
</tr>
<tr>
<td>emissive_intensity</td>
<td>Float</td>
<td>Emissive Intensity (kcd/m2)</td>
<td>Brightness simulating light emitting from the surface, making an object glow.</td>
</tr>
<tr>
<td>emissive_color</td>
<td>Color</td>
<td>Emissive Color</td>
<td>Tints the emissive color.</td>
</tr>
</tbody>
</table>

### Finding the Texture Name

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can specify texture names for nodes in the Script Canvas editor. The texture name is generally the path that starts at the top asset directory.

**To find the texture name**

1. In Lumberyard Editor, choose **Tools, Asset Browser**.
2. From the Asset Browser, navigate to the texture file.
3. Right-click the texture file and choose **Copy Path to Clipboard**. This copies the path to the file.
4. Delete parts of the path to get the texture name.

**Examples**

<table>
<thead>
<tr>
<th>Asset Location</th>
<th>Copied Path</th>
<th>Path to Delete</th>
<th>Texture Name to Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td><code>lumberyard_version\dev\SamplesProject\textures\defaults\white.tif</code></td>
<td><code>\SamplesProject\defaults\white.tif</code></td>
<td></td>
</tr>
<tr>
<td>Gem</td>
<td><code>lumberyard_version\dev\Gems\Rain\Assets\Textures\Rain\rainfall.tif</code></td>
<td><code>\Gems\Rain\Assets\Textures\Rain\rainfall.tif</code></td>
<td></td>
</tr>
<tr>
<td>Editor</td>
<td><code>lumberyard_version\dev\Editor\Plugins\ParticleEditorPlugin\defaults\feather01.tif</code></td>
<td><code>\Editor\Plugins\ParticleEditorPlugin\defaults\feather01.tif</code></td>
<td></td>
</tr>
<tr>
<td>Engine</td>
<td><code>lumberyard_version\dev\Engine\textures\skys\night\half_moon.tif</code></td>
<td><code>\Engine\textures\skys\night\half_moon.tif</code></td>
<td></td>
</tr>
</tbody>
</table>

**Writing Lua Scripts**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

**Topics**

- Learning Lua (p. 2684)
- Learning Lua in Lumberyard (p. 2684)
- Adding Lua Scripts to Component Entities (p. 2684)
- Basic Structure of a Component Entity Lua Script (p. 2686)
- Properties Table (p. 2687)
Learning Lua

For learning the Lua language itself, the lua.org website is a good place to start.

- **Official Lua Documentation** – Provides a central location for information about Lua, including a **Getting started** page.
- **Programming in Lua** – This text is a resource for getting started with Lua programming.
- **Lua 5.1 Reference Manual** – Provides a reference of all the functions that are available by default in Lua.

Learning Lua in Lumberyard

After you read through this tutorial on writing Lua scripts for the component entity system, learn more about using Lua in Lumberyard by consulting the following resources.

- For information on Lumberyard's built-in Lua editor, see Lua Editor (p. 2696).
- For sample Lua scripts, see the Lumberyard dev\SamplesProject\Scripts directory and its subdirectories.
- For information about Lua API operations in Lumberyard, see the Component Entity Lua API Reference (p. 2705).
- For information about the Lumberyard EBus, see Working with the Event Bus (EBus) system (p. 1851).

Adding Lua Scripts to Component Entities

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard makes it easy for you to add script functionality to your game entities by using the Lua Script component. The following steps show you how to do this in Lumberyard Editor.

**To add a Lua script to a component entity in Lumberyard Editor**

1. With the Entity Inspector view pane visible, select the entity in the viewport.
2. Click Add Component, and then open Scripting, Lua Script.
3. Scroll down to the **Scripting** section, and then click **Lua Script**.

4. A **Lua Script** component appears in the inspector. Use the file selection button (…) to select the Lua script from the file hierarchy that you want to use.
You can select either a .lua file (a text copy of the original), or a .luac file (a precompiled version of the script). The functionality should be the same. The precompiled version is preferable because it loads faster and is usually smaller. However, you can use .lua files if you experience any issues.

5. After the script is loaded, click Edit Script (j) to launch the Lua IDE and make changes to your script.

Basic Structure of a Component Entity Lua Script

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Scripts to be used as components contain a table (referred to as the script table), which provides the functionality for the script. In Lua, this table is treated like a class. The script table generally consists of the following:

- An optional Properties table within the script table. The Properties table provides an interface that you can use to customize the script behavior from the editor.
- An OnActivate() function that the engine calls when the entity that has the script is activated.
- An OnDeactivate() function called by the engine when the entity that has the script is deactivated.

The following example shows a skeleton script.

```lua
-- ScriptName.lua
local ScriptName =
{
    Properties =
    {
        -- Property definitions
```
For each Lua script component, Lumberyard creates a table called the entity table. The script table in the referenced script is the metatable for the entity table. Because of this relationship, when any method in the script is called, the self parameter (implicit in most cases) refers to the entity table.

The entity table then has the following properties and methods available to it:

- A `Properties` table, copied from the script table's `Properties` table. Default values are provided where appropriate.
- An `entityId` property, which contains an object of type `EntityId` that refers to the current entity.
- An `IsMaster` function, callable by the script, to check whether the currently executing script is on the primary node or a proxy node. This function is available only if the script component is network enabled.

**Built-in Types and Methods**

The Lumberyard engine provides a number of types and methods that are useful for making games. Many of the types and methods available are listed in the class view available in Lumberyard's Lua IDE. For more information on the class view, see Lua Editor (p. 2696).

**Properties Table**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The `Properties` table configures the editor interface for customizing the behavior of a script. With the properties table, you can modify numeric values, select states, and turn flags on and off. The table can even provide a reference to entities that your script can interact with.

The properties inside the `Properties` table are exposed to the editor. Properties outside the `Properties` table are private and not displayed in the editor.

The following example is a properties table from the **Controllable Chicken** sample level.

```lua
-- Example Properties Table
local ChickenMannequinControllerSM = {
    Properties = {
        MoveSpeed = { default = 3.0, description = "How fast the chicken moves.", suffix = " m/s" },
        RotationSpeed = { default = 360.0, description = "How fast (in degrees per second) the chicken can turn.", suffix = " deg/sec" },
    }
}
```
The result is the following Properties user interface in Lumberyard Editor:

The type that you provide as the default value determines how the property is appears in the editor user interface. You can further customize the representation of the property in the editor by specifying additional attributes in a table format. All property types support a description field that appears when you pause your mouse on the property name in the editor.

**Supported Types**

Properties can have the types described in this section.

**Boolean Values (True, False)**

The following examples are Boolean values.

```lua
DebugMovement = false,
AllowMovement = { default = true, description = "Allow or restrict movement of the object." },
```

In Lumberyard Editor, Boolean values are represented by a check box.

**Numeric Values (Integer or Floating Point Numbers)**

The following examples are numeric values.

```lua
Count = 5,
Velocity = { default = 1.0, suffix = "m/s", description = "Initial Velocity Of The Object" },
```
Distance = { default = 5.0, min = 2.0, max = 10.0, step = 2.0, suffix = "m", description = "The Distance An Object Can Travel In Meters" },

In Lumberyard Editor, numeric values are represented by an edit field with increase/decrease arrows. Numeric values can do the following:

- Provide a custom suffix to indicate units.
- Set minimum and maximum values.
- Provide a step value (how much the value increases or decreases when the user clicks the arrows on the right side of the edit field).

**Strings**

The following examples are strings.

DebugPrefix = "d_.",
Name = { default = "Default Name", description = "The name of the entity" },
StartingState = { "Idle", description = "Specify the starting state. Valid starting states are Idle and Fidget" },

In Lumberyard Editor, string values are represented by a text edit box.

**Reflected Classes**

You can use any class that is reflected to both the BehaviorContext and the EditContext as a property. A good example of this is the EntityId type, which references other entities.

```-- Entity Examples
ParentEntity = { default = EntityId(), description = "The Entity that this one will follow"},
Target = EntityId()
```

The editor representation is the default editor for the type reflected. For example, for EntityId, it's the entity reference picker. For most reflected types, it is a tree of the type's properties.

**Arrays**

Properties can contain resizable arrays of any of the types mentioned. To create a simple array, declare the default value as a keyless table of values. For example, the property definitions in the following code produce the properties shown in the image that follows.

```local ExampleScript = {
    Properties = {
        Speed = 4,
        ExampleArray = { default = { 1, 2, 3, 4 } },
    }
} return ExampleScript```
In the Entity Inspector, you can use the green + and red X icons to add and remove entries in real time. You can also use EntityId() to make the array elements entity references, as in the following example from the `lumberyard_version\dev\Gems\LyShineExamples\Assets\UI\Scripts\LyShineExamples\Image\ImageFillTypes.lua` file.

```lua
local ImageFillTypes = {
  Properties = {
    FilledImages = { default = { EntityId(), EntityId(), EntityId(), EntityId() } },
    Dropdowns = { default = { EntityId(), EntityId(), EntityId(), EntityId() } },
    RadialStartAngleSlider = { default = EntityId() },
    SpriteRadioButtonGroup = { default = EntityId() },
  },
}
...
```

### Grouping Properties

The following code example shows how you can use variables within the `Properties` table to expose named groupings of properties.

```lua
local Test = {
  Properties = {
    Movement = {
      TopSpeed = 4,
      Acceleration = 2,
      TurnSpeed = 12,
    },
    Combat = {
      ProjectileDamage = 50,
      RateOfFire = 3,
      AmmoCapacity = 12,
    },
  },
}
return Test
```

When rendered in the UI, the property groupings and properties are sorted alphabetically irrespective of their order in the code.
Attributes

You can add attributes to a property by placing them alongside the default value in a property table. Attribute keys are not case sensitive. The following common attributes can be added to any property.

Common Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>is the text of the tool tip for the property.</td>
</tr>
<tr>
<td>UI</td>
<td>Specifies (overrides) the UI handler that the property uses.</td>
</tr>
</tbody>
</table>

Network Binding Properties

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For network binding features to function, you must have a Creating a NetBindable Component (p. 2055).

Properties

You can configure networking binding for properties by adding the netSynched table to the description of the variable inside of the Properties table.

```lua
local ExampleScript = {
    Properties = {
        Speed = {
            default = 0, -- Supports numbers, strings, booleans, and nils for net bindings.
            min = 0,
            max = 100,
            step = 1,
            description = "Speed in m/s for the ...",
```
Network Binding Properties

-- If this table is missing, it is assumed the value is not networked.

netSynched =
{
   -- Optional fields
   OnNewValue = <function> -- OnNewValue is called whenever the property has a
   -- new value. OnNewValue accepts one
   parameter, which
   -- is the entity table for the instance that
   changed.
   -- The following flags are mainly here for debugging and profiling
   Enabled = true          -- Controls whether the field is network enabled.
   -- missing, assumes true.
   ForceIndex = [1..32]    -- Profiling helper tool to force a property to use
   -- specific DataSet to make understanding what data
   -- being used where easier.
}

return ExampleScript

After you add networking to a property, any changes to the property are reflected across the network.

RPCs

Exposing RPCs to scripts involves creating a new table inside of the component table, but outside of the
properties table, as shown in the following example.

local ExampleScript = {
   Properties = {
      -- ...
   },

   -- Table of remote procedure calls (RPCs) that the script wants to implement.
   NetRPCs =
   {
      RPCNoParam = {
         OnMaster = <function> -- The function to be called on the Primary Script.
         -- The function should return a bool value that
         -- indicates whether or not proxy components can
         -- execute the RPC on themselves. Required.
         OnProxy = <function> -- The function to be called on the Proxy Script.
         -- This function is optional and can be excluded if
         -- the master never allows proxies to execute the
         function call.
      }
   }

return ExampleScript

You can invoke the RPC just like any other function. There is no need to specify the OnMaster/OnProxy
from the calling script. For example, you can call RPCs as in the following example.

self.NetRPCs.RPCNoParam()
self.NetRPCs.RPCParam(1.0)
Using EBuses in Lua

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Components provide interfaces that allow scripts to send them information and receive notifications when certain actions take place. Communication is established by creating two different objects in Lua: senders and handlers. A sender or a handler is an interface to an EBus (p. 1851), a communication system used extensively in the Lumberyard Engine. When a sender is created, it can call functions, which in turn send information to a component. When a handler is created, the component calls certain functions that the Lua script defines. These senders and handlers are created with an entity ID. You can use the entity ID to communicate with components that are attached to entities other than the one the script itself is running on. The main script table always provides a field called entityId that contains the ID of the entity to which the script is attached. Other entity IDs can be passed to the script through the Properties interface.

Order of Component Activation

Keep in mind the following points regarding the order of activation of Lua components:

- Lua components are activated after all C++ components have been activated.
- If an entity has multiple Lua components, there is no guarantee regarding which Lua component is activated first.

Communicating with Components

When a Lua script creates a handler object, it notifies a component attached to an entity that it should call the script handler functions when certain events occur. For example, in the first sample below, the script creates a Spawner (p. 859) notification bus handler when OnActivate() is called. This tells the spawner component attached to the entity that has the script to call the OnSpawnBegin(), OnSpawnEnd(), and OnEntitySpawned() functions when the spawner instantiates a new dynamic slice (p. 524). Subsequently, the handler is explicitly disconnected and set back to nil in the OnDeactivate function. This ensures that processing time is not wasted when the entity attached to the script isn't active. As long as the entity is active, these functions are called by the spawner component at the appropriate time.

The following code example shows a spawner component handler.

```lua
local SpawnerScriptSample = { }

function SpawnerScriptSample:OnActivate()
    -- Register our handlers to receive notification from the spawner attached to this entity.
    if( self.spawnerNotiBusHandler == nil ) then
        self.spawnerNotiBusHandler = SpawnerComponentNotificationBus.CreateHandler(self,
                        self.entityId)
    end
end

-- This handler is called when we start spawning a slice.
function SpawnerScriptSample:OnSpawnBegin(sliceTicket)
    -- Do something so we know if/when this is being called
    Debug.Log("Slice Spawn Begin")
end

-- This handler is called when we're finished spawning a slice.
```
function SpawnerScriptSample:OnSpawnEnd(sliceTicket)
    -- Do something so we know if/when this is being called
    Debug.Log("Slice Spawn End")
end

-- This handler is called whenever an entity is spawned.
function SpawnerScriptSample:OnEntitySpawned(sliceTicket, entityId)
    -- Do something so we know if/when this is being called
    Debug.Log("Entity Spawned: " .. tostring(entityId) )
end

function SpawnerScriptSample:OnDeactivate()
    -- Disconnect our spawner notification
    if self.spawnerNotiBusHandler ~= nil then
        self.spawnerNotiBusHandler:Disconnect()
        self.spawnerNotiBusHandler = nil
    end
end

return SpawnerScriptSample

Noncomponent Notifications

Some event buses that are available to Lua are not associated with components. For example, the system's tick bus (p. 1019) is not a component bus and does not require an entity ID. It provides both the amount of time that has passed since the last engine tick and the current time point. To gain access to this information, write a script that implements the `OnTick()` function and creates the handler. The handler receives notifications from the system whenever the engine ticks.

The following example shows how to register with the tick bus.

```lua
local TestScript = { }
function TestScript:OnActivate()
    -- Inform the tick bus that you want to receive event notifications
    self.tickBusHandler = TickBus.CreateHandler(self)
    self.tickBusHandler:Connect()
end

-- This callback is called every frame by the tick bus after this entity activates
function TestScript:OnTick(deltaTime, timePoint)
    -- Add script to be executed every frame here...
    end

function TestScript:OnDeactivate()
    -- Inform the tick bus that you no longer want to receive notifications
    self.tickBusHandler:Disconnect()
end

return TestScript
```

**Note**

Instead of calling `CreateHandler` and then calling `Connect` on the handler, you can use the Lua shortcut function `TickBus.Connect`. The `Connect` function uses the following syntax to create a handler and automatically connect the handler to the bus.

```
handler = TickBus.Connect(handlerTable[, connectionId])
```

Sending Events to a Component

In addition to receiving notifications from components, a script must sometimes exercise control over components. Control is accomplished by sending events to components using the `Event` table and
calling the functions implemented on it. In the example script that follows, the **Spawner** (p. 859) component is sent an event that tells the component to spawn a dynamic slice by calling the `Spawn()` function. The first argument to an `Event` function is always the ID of the listener that you send the event to; the remaining arguments follow.

The following example shows how to send EBus events.

```lua
local SpawnerScript = { }
function SpawnerScript:OnActivate()
    SpawnerComponentRequestBus.Event.Spawn(self.entityId)
end
return SpawnerScript
```

You can request information from some event sending functions that return values. The next example script uses a `TransformBus` to get the current local transform of the entity and uses the `GetLocalTM()` function, which returns a transform object. This object is stored in a variable in the main script table. `TransformBus` is used again to reset the transform of the object to the identity.

The following example shows how to use the transform bus.

```lua
function samplescript:OnActivate()
    -- Retrieve the object's local transform and store it for later use
    self.myOldTransform = TransformBus.Event.GetLocalTM(self.entityId)

    -- Reset the object's local transform to the identity matrix
    TransformBus.Event.SetLocalTM(self.entityId, Transform.CreateIdentity())
end
```

**Communicating with Components Attached to Other Entities**

You can also send events and create handlers to communicate with components that are attached to other entities. The following example defines a parent entity in the properties table and requests its transform. This allows it to set its transform to that of another entity.

```lua
local ParentScriptSample = { }
    Properties = {
        ParentEntity = {default = EntityId()}
    }
}

function ParentScriptSample:OnActivate()
    if self.Properties.ParentEntity:IsValid() then
    end
end

function ParentScriptSample:OnEntityActivated()
    TransformBus.Event.SetLocalTM(self.entityId, parentTransform)
end
return ParentScriptSample
```

**Important**

If you have a Lua script that is attached to an entity that needs to get information from another entity, your script must subscribe to the target entity's `OnEntityActivated` event. Your script should wait for the target entity to be activated before requesting the relevant information. Otherwise, your script might return nil.
Using AZStd::vector and AZStd::array

Vectors and arrays in Lua behave very similarly to tables, with a few limitations. Both vector and array have the following features.

Length Operator #

You can obtain the length of a collection by prefixing the name of the collection with the length operator #, as in the following example.

```
#myCollection
```

Indexing []

To obtain the elements in a collection, use indexing in square brackets as the following syntax shows. Indexing is 1 based, just like Lua tables.

```
myCollection[index]
```

Vector also has the following methods for mutating the collection.

**push_back**

Use the `push_back` method to append elements to the vector, as in the following example.

```
myCollection:push_back(5)
```

**pop_back**

Use the `pop_back` method to remove the last element of the vector, as in the following example.

```
myCollection:pop_back()
```

**clear**

Use the `clear` method to remove all elements from the vector, as in the following example.

```
myCollection:clear()
```

Using AZStd::any

You can pass any Lua primitive type excluding tables to any bus or function that takes AZStd::any as a parameter (for example, GameplayNotificationBus::OnEventBegin). You can also pass any type reflected from C++ (for example, vectors or EntityId values). There is no syntax required to pass a value as an any—just call the bus or function.

The following example shows the use of AZStd::any.

```
GameplayNotificationBus.Broadcast.OnEventBegin(self.eventId, "The value I'd like to pass to the handler")
```

Lua Editor

This feature is in preview release and is subject to change.
Lumberyard Lua Editor (Lua IDE) offers an intuitive integrated development environment (IDE) that makes it easy to author, debug, and edit Lua scripts when you create or extend your game. Lua Editor is a standalone application, but can be opened directly from Lumberyard Editor using the Tools menu.

**Tutorial: Using Lua Editor for Debugging with Lumberyard Editor**

This tutorial shows you how to use Lumberyard Editor to create a sample level in the **SamplesProject** project with a component entity that contains a Lua script component. You will learn how to open the script in Lua Editor and perform some sample debugging steps on the script.

**Debugging Lua scripts using Lua Editor**

1. Set the **SamplesProject** as the default project using Project Configurator.
2. In Lumberyard Editor, create a new level by performing one of the following steps:
   - In the **Welcome to Lumberyard Editor** window, click **New level**
   - Click **File**, **New**
   - Press **Ctrl+N**
3. In the **New Level** dialog box, give the level a name, and then click **OK**.
4. In **Asset Browser**, expand **SamplesProject**, **Objects**, and **SamplesAssets**.
5. Drag **mover_display_smooth.cgf** to the perspective viewport.
6. In **Entity Inspector**, click **Add Component**, and then choose **Scripting**, **Lua Script**.
7. In the **Entity Inspector** window, locate the **Lua Script** component, and then click the **Pick Lua Script** button next to the empty **Script** field.
8. In the **Pick Lua Script** window, expand **SamplesProject**, **Scripts**, and **Components**.
9. Click **ConstantRotation.lua**, and then click **OK**.
10. In the **Lua Script** component, click the **Open in Lua Editor** button to launch Lua Editor.

Because the debugging functionality is enabled through network sockets, you must connect Lua Editor to the target that is running the script before you can debug. In this tutorial, you connect to Lumberyard Editor.
Note
Connection is facilitated by GridHub (p. 1954), which is Lumberyard's central connection hub for debugging. GridHub starts automatically when Lua Editor is started and must be running in the background for Lua Editor to find targets it can connect to.

11. In the Lua Editor toolbar, click **Target: None**, and then click **Editor(ID)** to connect to Lumberyard Editor.

Note
You may need to expand the Lua Editor window to see the buttons on the Lua Editor toolbar for the next few steps.

12. In the Lua Editor toolbar, leave **Context** setting at **Default** for the debugging context. The default setting is good for debugging component entity scripts such as the one in this tutorial. The **Cry** context option is for debugging legacy scripts such as those associated with Cry entities or the Game SDK.

13. The **Debugging** icon turns green to show that Lua Editor and Lumberyard Editor are connected:

Click **Classes** in the **Class Reference** to show the available Lua libraries. You can do the same for **EBuses** and **Globals**.
Note
The class reference feature is active only for the default context and component entity scripts. This feature is not active in the Cry context, which exists only for backward compatibility.

After you connect, you can pause the execution of a given script by setting breakpoints.

14. In the Lua Editor toolbar, click the **Breakpoints** icon ![Breakpoints icon](image) to show the **Breakpoints** window.

15. In Lua Editor, click one or more line numbers in the `constantrotation.lua` script to set one or more breakpoints. As you add breakpoints, the line number and script path for each are added to the **Breakpoints** window.

16. In Lumberyard Editor, press **Ctrl+G** to run the game, or click the **Simulate** icon at the bottom of the viewport to enable game simulation and run scripts. Lua Editor opens with a yellow marker stopped on the first breakpoint that it encounters.
When execution is halted at a breakpoint, more information becomes available in the Lua Locals, Stack, and Watched Variables panes.

17. Click the Stack icon to show the Stack window.

18. Click the Lua Locals icon to show local Lua variables.

19. Click Watched Variables icon to open the Watched Variables window, where you can specify variables to watch.

20. Press F11 a few times to step through the code. Note how the contents of the Stack, Lua Locals, and Watched Variables windows change.

   **Tip**
   For greater convenience, you can float or dock these windows.

21. To detach from debugging, click Debugging.

22. In Lumberyard Editor, Press Esc to stop the game.

**Options Available While Debugging**

The following table summarizes common options available while debugging.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Action</th>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="icon" alt="run_in_editor" /></td>
<td>Run in Editor</td>
<td>Alt+F5</td>
<td>Run in Lumberyard Editor.</td>
</tr>
<tr>
<td><img src="icon" alt="run_on_target" /></td>
<td>Run on Target</td>
<td>Ctrl+F5</td>
<td>Send script to the connected target and run it.</td>
</tr>
<tr>
<td><img src="icon" alt="run_continue" /></td>
<td>Run/Continue</td>
<td>F5</td>
<td>Run or continue running the current script.</td>
</tr>
<tr>
<td><img src="icon" alt="step_into" /></td>
<td>Step Into</td>
<td>F11</td>
<td>Step into the function called on the current line.</td>
</tr>
<tr>
<td><img src="icon" alt="step_out" /></td>
<td>Step Out</td>
<td>Shift+F11</td>
<td>Step out of the called function.</td>
</tr>
<tr>
<td><img src="icon" alt="step_over" /></td>
<td>Step Over</td>
<td>F10</td>
<td>Step over the function called on the current line.</td>
</tr>
<tr>
<td>Icon</td>
<td>Action</td>
<td>Keyboard Shortcut</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>🕵️‍♂️</td>
<td>Toggle Breakpoint</td>
<td>F9</td>
<td>Enable or disable a breakpoint on the current line.</td>
</tr>
</tbody>
</table>

**Maintaining Separate Search Results**

In addition to the usual search capabilities, the **Find** feature can display the results of four different searches separately.

**To maintain separate search results**

1. Click the **Find** icon 🕵️‍♂️ or press **Ctrl+F** to perform searches in the currently open file, or in all open files.

2. Before starting a search, choose **Find 1**, **Find 2**, **Find 3**, or **Find 4** to choose the window in which you want to see the results. You can maintain the results of four searches separately in the tabbed windows. The search results in the other windows remain unchanged.
3. To go directly to the line in the code which a search result was found, double-click the line in the search results.

   **Note**
   In Lua Editor Preview, the line number shown in the **Find Results** window and the line number in the script pane differ by one.

   **Tip**
   For convenience, you can also dock or float the **Find Results** window.

**Editing**

Lua Editor can open multiple scripts at the same time. Each script has its own tab in the editor. The editor provides a standard set of capabilities for text editing but also includes useful features for editing source code.

The following table summarizes the options available while editing and debugging.

<table>
<thead>
<tr>
<th>Action</th>
<th>Keyboard Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment selected block</td>
<td>Ctrl+K</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Find</td>
<td>Ctrl+F</td>
</tr>
<tr>
<td>Find in open files</td>
<td>Ctrl+Shift+F</td>
</tr>
<tr>
<td>Find next</td>
<td>F3</td>
</tr>
<tr>
<td>Fold source functions</td>
<td>Alt+0</td>
</tr>
<tr>
<td>Go to line</td>
<td>Ctrl+G</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Quick find local</td>
<td>Ctrl+F3</td>
</tr>
<tr>
<td>Quick find local reverse</td>
<td>Ctrl+Shift+F3</td>
</tr>
<tr>
<td>Redo</td>
<td>Ctrl+Y</td>
</tr>
<tr>
<td>Replace</td>
<td>Ctrl+R</td>
</tr>
<tr>
<td>Replace in open files</td>
<td>Ctrl+Shift+R</td>
</tr>
<tr>
<td>Select all</td>
<td>Ctrl+A</td>
</tr>
<tr>
<td>Select to brace¹</td>
<td>Ctrl+Shift+]</td>
</tr>
</tbody>
</table>
### Debugging Lua Scripts

Lumberyard provides Lua scripts with several functions to make debugging easier.

#### Logging to the Console

To print text to Lumberyard Editor and the game console, use the `Debug.Log()` function.

The following example shows the use of the `Debug.Log()` function.

```lua
local LoggingTest = { }
function LoggingTest:OnActivate()
    componentName = "MyComponent"
    Debug.Log(componentName .. " has been activated.")
end
```

---

**Perforce Integration**

Lua Editor includes Perforce integration features. When you open a file from your Perforce environment, Lua Editor displays the file's status in the top right of the text editing window.

The Source Control menu offers Check Out/Check In functionality.

---

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Using an Assert to Detect Potential Issues

You can use the `assert` function to display an error message in the console when conditions are detected that might result in an execution fault. The `assert` function takes two arguments: a condition that evaluates to true or false, and a message to display if the condition is false.

The following example shows the use of the `assert` function.

```lua
function SampleScript:DoStuff()
    -- This value should never be negative
    assert( self.positiveValue >= 0, "Expected a positive value! Got: " ..
            self.positiveValue )
end

-- Console output when the value of self.positiveValue is -5:
-- [Error] Lua error (2 - [string "q:/lyengine/branches/systems/dev/samplespro...":61: Expected a positive value! Got: -5) during call samplescript:DoStuff

Communicating Errors

You can use the `Debug.Error()` function to display an error in the console and halt execution of the current script function. This does not halt all execution of the script. If you have active handlers, they can still be called when the engine posts notifications. The `Debug.Error()` function takes arguments similar to the `Debug.Assert` function: a condition and a message. The message is displayed in bright red and execution halts only if the condition is false.

The following example shows the use of the `Debug.Error()` function.

```lua
function SampleScript:CheckAndError()
    -- This value should never be negative
    Debug.Error( self.positiveValue >= 0, "Detected a negative value: " ..
                      self.positiveValue )
end

-- Console output when the value of self.positiveValue is -5:
-- [Error] Error on argument 0: Detected a negative value: -5

Displaying a Warning When User Attention Is Required

A script condition can occur that does not adversely affect the execution of the script but might be useful for the user to know about. The `Debug.Warning()` function uses arguments similar to those of the `Error` and `Assert` functions but just displays an orange warning message in the console. It does not halt execution.

The following example shows the use of the `Debug.Warning()` function.

```lua
function SampleScript:CheckValue()
    -- This value should probably never be negative
    Debug.Warning( self.positiveValue >= 0, "Detected a negative value: " ..
                      self.positiveValue )
end

-- Console output when the value of self.positiveValue is -5:
-- [Warning] Warning on argument 0: Detected a negative value: -5

Version 1.28
The Lua Environment (Advanced)

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

By default, the Lumberyard component entity Lua environment is a single Lua environment (or `lua_State`). This environment is bound to the `BehaviorContext` that is owned by the `ComponentApplication`. Because of this, it has access to all API operations that are reflected on startup.

Adding Other VMs

You may add more `ScriptContext` instances using the `ScriptSystemBus` (either call `AddContextWithId`, or create your own and call `AddContext`). If you want your new context to be available for debugging, you must register it with `ScriptDebugAgentBus::RegisterContext`.

Reusing Code

Lua provides the capability to load and execute scripts from other Lua files using the built-in Lua `require` function. It's important to note that this function requires a special path format. The file path is delimited by periods instead of slashes, has no `.lua` file name extension, and is relative to the Lumberyard assets directory. For example, if you want to use the `require` function to give your scripts some common functionality from the project's `Scripts` directory, you can use code similar to the following example.

```lua
local library = require("Scripts.MyLibraryFile")
```

Component Entity Lua API Reference

This documentation is preliminary and subject to change.

BehaviorTreeComponentRequestBus

Represents a request submitted by a user of the current component.

StartBehaviorTree

Starts an inactive behavior tree associated with the current entity.

Syntax
void BehaviorTreeComponent::StartBehaviorTree()

StopBehaviorTree

Stops an active behavior tree associated with the current entity.

Syntax

void BehaviorTreeComponent::StopBehaviorTree()

GetVariableNameCrcs

Gets a list of cyclic redundancy check values for variable names.

Syntax

AZStd::vector<AZ::Crc32> GetVariableNameCrcs()

Returns: A list of the 32-bit cyclic redundancy check values for all variable names.

Return Type: AZStd::vector

Default Return: s_defaultEmptyVariableIds

GetVariableValue

Gets the value for the specified variable name CRC-32 checksum.

Syntax

bool GetVariableValue(AZ::Crc32 variableNameCrc)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableNameCrc</td>
<td>AZ::Crc32</td>
<td>The CRC-32 checksum for the variable name.</td>
</tr>
</tbody>
</table>

Returns: true if successful; otherwise, false.

Return Type: bool

Default Return: false

SetVariableValue

Set the value associated with a variable.

Syntax

void SetVariableValue(AZ::Crc32 variableNameCrc, bool newValue)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableNameCrc</td>
<td>AZ::Crc32</td>
<td>The CRC-32 checksum for the variable name.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>newValue</td>
<td>bool</td>
<td>The new value for the variable.</td>
</tr>
</tbody>
</table>

**NavigationComponentRequestBus**

Requests serviced by the navigation component.

**FindPathToEntity**

Creates a path finding request to navigate towards the specified entity.

**Syntax**

```lua
PathfindRequest::NavigationRequestId FindPathToEntity(AZ::EntityId entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AZ::EntityId</td>
<td>Request EntityId of the entity we want to navigate towards.</td>
</tr>
</tbody>
</table>

**Returns:** A unique identifier to the pathfinding request.

**Return Type:** PathfindRequest::NavigationRequestId

**Default Return:** PathfindResponse::kInvalidRequestId

**Stop**

Stops all pathfinding operations for the specified requestId. The ID is used to make sure that the request being cancelled is the request that is currently being processed. If the specified requestId is different from the ID of the current request, the stop command can be safely ignored.

**Syntax**

```lua
void Stop(PathfindRequest::NavigationRequestId requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>PathfindRequest::NavigationRequestId</td>
<td>ID of the request that is being cancelled.</td>
</tr>
</tbody>
</table>

**NavigationComponentNotificationBus**

Notifications sent by the Navigation component.

**OnSearchingForPath**

Indicates that the pathfinding request has been submitted to the navigation system.

**Syntax**

```lua
void OnSearchingForPath(PathfindRequest::NavigationRequestId requestId)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>PathfindRequest::NavigationRequestId</td>
<td>ID of the request for the path that is being searched.</td>
</tr>
</tbody>
</table>

**OnTraversalStarted**

Indicates that traversal for the indicated request has started.

**Syntax**

```c
void OnTraversalStarted(PathfindRequest::NavigationRequestId requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>PathfindRequest::NavigationRequestId</td>
<td>ID of the request for which traversal has started.</td>
</tr>
</tbody>
</table>

**OnTraversalInProgress**

Indicates that traversal for the indicated request has started.

**Syntax**

```c
void OnTraversalInProgress(PathfindRequest::NavigationRequestId requestId, float distanceRemaining)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>PathfindRequest::NavigationRequestId</td>
<td>ID of the request for which traversal is in progress.</td>
</tr>
<tr>
<td>distanceRemaining</td>
<td>float</td>
<td>The remaining distance in the current path.</td>
</tr>
</tbody>
</table>

**OnTraversalComplete**

Indicates that traversal for the indicated request has completed successfully.

**Syntax**

```c
void OnTraversalComplete(PathfindRequest::NavigationRequestID requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>PathfindRequest::NavigationRequestId</td>
<td>ID of the request for which traversal has finished.</td>
</tr>
</tbody>
</table>

**OnTraversalCancelled**

Indicates that traversal for the indicated request was cancelled before it could be successfully completed.

**Syntax**

```c
```
void OnTraversalCancelled(PathfindRequest::NavigationRequestId requestId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>PathfindRequest::NavigationRequestId</td>
<td>ID of the request for which traversal was cancelled.</td>
</tr>
</tbody>
</table>

**NavigationSystemRequestBus**

Requests serviced by the navigation system component. This currently contains the single function `RayCast`.

**RayCast**

Creates a path finding request to navigate towards the specified entity.

**Syntax**

```cpp
definition RayCast(const AZ::Vector3& begin, const AZ::Vector3& direction, float maxDistance) { return NavRayCastResult() }
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td>Vector3</td>
<td>The origin of the ray.</td>
</tr>
<tr>
<td>direction</td>
<td>Vector3</td>
<td>The direction for the ray to travel.</td>
</tr>
<tr>
<td>maxDistance</td>
<td>float</td>
<td>The maximum distance the ray travels.</td>
</tr>
</tbody>
</table>

**Returns:** A `NavRayCastResult`.

`NavRayCastResult` has the following structure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_collision</td>
<td>Boolean</td>
<td>Returns true if there was a collision. The default is false.</td>
</tr>
<tr>
<td>m_position</td>
<td>Vector3</td>
<td>The position of the hit in world space. The default is AZ::Vector3::CreateZero().</td>
</tr>
<tr>
<td>m_meshId</td>
<td>NavigationMeshId</td>
<td>The mesh ID of the navigation mesh hit. This is callable from Lua script. The default is 0.</td>
</tr>
</tbody>
</table>

**AttachmentComponentRequestBus**

Messages serviced by the AttachmentComponent. The AttachmentComponent lets an entity "stick" to a particular bone on a target entity.

**Attach**

Change the attachment target. The entity will detach from any previous target.
Syntax

```cpp
void Attach(AZ::EntityId targetId, const char* targetBoneName, const AZ::Transform& offset)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetId</td>
<td>AZ::EntityId</td>
<td>Specifies the ID of the entity to attach to.</td>
</tr>
<tr>
<td>targetBoneName</td>
<td>char</td>
<td>Specifies the bone on the target entity to attach to. If the target bone is not found, then attach to the target entity's transform origin.</td>
</tr>
<tr>
<td>offset</td>
<td>AZ::Transform</td>
<td>The attachment's offset from the target.</td>
</tr>
</tbody>
</table>

**Detach**

Detaches an entity from its target.

**Syntax**

```cpp
void Detach()
```

**SetAttachmentOffset**

Update an entity's offset from its target.

**Syntax**

```cpp
void SetAttachmentOffset(const AZ::Transform& offset)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>AZ::Transform</td>
<td>The offset from the target.</td>
</tr>
</tbody>
</table>

**AttachmentComponentNotificationBus**

This EBus interface handles events emitted by the AttachmentComponent. The AttachmentComponent lets an entity "stick" to a particular bone on a target entity.

**OnAttached**

The entity has attached to the target.

**Syntax**

```cpp
void OnAttached(AZ::EntityId targetId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetId</td>
<td>AZ::EntityId</td>
<td>The target being attached to.</td>
</tr>
</tbody>
</table>
OnDetached

The entity is detaching from the target.

Syntax

void OnDetached(AZ::EntityId targetId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetId</td>
<td>AZ::EntityId</td>
<td>The target being detached from.</td>
</tr>
</tbody>
</table>

CharacterAnimationRequestBus

General character animation requests serviced by the CharacterAnimationManager component.

SetBlendParameter

Sets a custom blend parameter.

Syntax

void SetBlendParameter(AZ::u32 blendParameter, float value)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blendParameter</td>
<td>AZ::u32</td>
<td>Corresponds to EMotionParamID.</td>
</tr>
<tr>
<td>value</td>
<td>float</td>
<td>The value to set.</td>
</tr>
</tbody>
</table>

SetAnimationDrivenMotion

Enables or disables animation-driven root motion.

Syntax

void SetAnimationDrivenMotion(bool useAnimDrivenMotion)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useAnimDrivenMotion</td>
<td>bool</td>
<td>Specify true to enable animation-driven root motion; false to disable.</td>
</tr>
</tbody>
</table>

MannequinRequestsBus

Services provided by the Mannequin component.

QueueFragment

Queues the specified Mannequin fragment.
Syntax

```plaintext
FragmentRequestId QueueFragment(int priority, const char* fragmentName, const char* fragTags, bool isPersistent)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>int</td>
<td>Specifies priority. A higher number means higher priority</td>
</tr>
<tr>
<td>fragmentName</td>
<td>char</td>
<td>Name of the fragment to be played.</td>
</tr>
<tr>
<td>fragTags</td>
<td>char</td>
<td>Fragment tags to be applied.</td>
</tr>
<tr>
<td>isPersistent</td>
<td>bool</td>
<td>Specifies persistence.</td>
</tr>
</tbody>
</table>

**Returns**: A request ID that can be used to identify and make modifications to the request.

**Return Type**: FragmentRequestId

**Default Return**: MannequinRequests::s_invalidRequestId

### PauseAll

Pauses all actions being managed by the current Mannequin component.

**Syntax**

```plaintext
void PauseAll()
```

### ResumeAll

Resumes all actions being managed by the current Mannequin component.

**Syntax**

```plaintext
void ResumeAll(IActionController::EResumeFlags resumeFlag)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resumeFlag</td>
<td>IActionController::EResumeFlags</td>
<td>Indicates how the animations are to be resumed. See the EResumeFlags enum for possible values.</td>
</tr>
</tbody>
</table>

```plaintext
eenum EResumeFlags
{
    ERF_RestartAnimations = BIT(0),
    ERF_RestoreLoopingAnimationTime = BIT(1),
    ERF_RestoreNonLoopingAnimationTime = BIT(2),
    ERF_Default = ERF_RestartAnimations | ERF_RestoreLoopingAnimationTime | ERF_RestoreNonLoopingAnimationTime
};
```
SetTag

Sets the specified tag for the action controller.

**Syntax**

```cpp
void SetTag(const char* tagName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagName</td>
<td>char</td>
<td>The name of the tag to set.</td>
</tr>
</tbody>
</table>

ClearTag

Clears the specified tag for the action controller.

**Syntax**

```cpp
void ClearTag(const char* tagName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagName</td>
<td>char</td>
<td>The name of the tag to be cleared.</td>
</tr>
</tbody>
</table>

SetGroupTag

Sets a tag in the specified group.

**Syntax**

```cpp
void SetGroupTag(const char* groupName, const char* tagName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>char</td>
<td>The name of the group.</td>
</tr>
<tr>
<td>tagName</td>
<td>char</td>
<td>The name of the tag.</td>
</tr>
</tbody>
</table>

ClearGroup

Clears tags for the indicated group.

**Syntax**

```cpp
void ClearGroup(const char* groupName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>char</td>
<td>The name of the group.</td>
</tr>
</tbody>
</table>
SetScopeContext

Sets the scope context for the current animation controller.

Syntax

```c
void SetScopeContext(const char* scopeContextName, const AZ::EntityId entityId, const char* animationDatabaseName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeContextName</td>
<td>char</td>
<td>Name of the scope context that the animation database (.adb) file is to be attached to.</td>
</tr>
<tr>
<td>entityId</td>
<td>AZ::EntityId</td>
<td>Reference to an entity whose character instance will be bound to this scope context.</td>
</tr>
<tr>
<td>animationDatabaseName</td>
<td>char</td>
<td>The path to the animation database file.</td>
</tr>
</tbody>
</table>

ClearScopeContext

Clears the specified scope context.

Syntax

```c
void ClearScopeContext(const char* scopeContextName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeContextName</td>
<td>char</td>
<td>Name of the scope context that is to be cleared.</td>
</tr>
</tbody>
</table>

StopRequest

Stops the actions associated with the specified request.

Syntax

```c
void StopRequest(FragmentRequestId requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>Specifies the ID of the request for which actions should be stopped.</td>
</tr>
</tbody>
</table>

GetRequestStatus

Retrieves the status of the specified request.

Syntax

```c
IAction::EStatus GetRequestStatus(FragmentRequestId requestId)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>The ID of the request to retrieve status for.</td>
</tr>
</tbody>
</table>

**Returns:** The status of the request.

**Return Type:** IAction::EStatus

**Default Return:** IAction::EStatus::None

**ForceFinishRequest**

Forces the actions associated with the specified request to finish.

**Syntax**

```cpp
void ForceFinishRequest(FragmentRequestId requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>The ID of the request.</td>
</tr>
</tbody>
</table>

**SetRequestSpeedBias**

Sets the speed bias for the actions associated with the specified request.

**Syntax**

```cpp
void SetRequestSpeedBias(FragmentRequestId requestId, float speedBias)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>The request ID.</td>
</tr>
<tr>
<td>speedBias</td>
<td>float</td>
<td>The speed bias for this animation</td>
</tr>
</tbody>
</table>

**GetRequestSpeedBias**

Gets the speed bias for the actions associated with the specified request.

**Syntax**

```cpp
float GetRequestSpeedBias(FragmentRequestId requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>The ID of the request.</td>
</tr>
</tbody>
</table>

**Returns:** The speed bias for the indicated request.

**Return Type:** float
**Default Return:** -1

**SetRequestAnimWeight**
Sets the animation weight for the actions associated with the specified request.

**Syntax**

```lua
void SetRequestAnimWeight(FragmentRequestId requestId, float animWeight)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>The ID of the request.</td>
</tr>
<tr>
<td>animWeight</td>
<td>float</td>
<td>The weight for the animation.</td>
</tr>
</tbody>
</table>

**GetRequestAnimWeight**
Gets the animation weight for the actions associated with the specified request.

**Syntax**

```lua
float GetRequestAnimWeight(FragmentRequestId requestId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>FragmentRequestId</td>
<td>The ID of the request.</td>
</tr>
</tbody>
</table>

**Returns:** The animation weight for the indicated request.

**Return Type:** float

**Default Return:** -1

**SimpleAnimationComponentRequestBus**

Services provided by the Simple Animation component. The Simple Animation component provides basic animation functionality for the entity. If the entity has a mesh component with a skinned mesh attached (a .chr or .cdf file), the Simple Animation component will provide a list of all valid animations specified in the associated .chrparams file. The Simple Animation component does not provide interaction with Mannequin and should be used for light-weight environment or background animation.

**StartDefaultAnimations**

Plays the default animations along with default looping and speed parameters that were set up as a part of the current component. Components allow for multiple layers to be set up with defaults. The `StartDefaultAnimations` method starts the playback of all the default animations of the component.

**Syntax**

```lua
SimpleAnimationComponentRequests::Result StartDefaultAnimations()
```

**Returns:** A Result indicating whether the animations were started successfully.
Return Type: SimpleAnimationComponentRequests::Result

Default Return: SimpleAnimationComponentRequests::Result::Failure

StartAnimation

Starts playback of the animation of the specified animatedLayer.

Syntax

```cpp
SimpleAnimationComponentRequests::Result StartAnimation(const AnimatedLayer& animatedLayer)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>animatedLayer</td>
<td>AnimatedLayer</td>
<td>A layer configured with the animation that is to be played on it.</td>
</tr>
</tbody>
</table>

Returns: A Result indicating whether the animation was started.

Return Type: SimpleAnimationComponentRequests::Result

Default Return: SimpleAnimationComponentRequests::Result::Failure

StartAnimationByName

Plays the animation with the specified name.

Syntax

```cpp
Result StartAnimationByName(const char* name, AnimatedLayer::LayerId layerId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>char</td>
<td>The name of the animation to play.</td>
</tr>
<tr>
<td>layerId</td>
<td>AnimatedLayer::LayerId</td>
<td>The layer in which to play the animation</td>
</tr>
</tbody>
</table>

Returns: A Result indicating whether the animation was started.

Return Type: SimpleAnimationComponentRequests::Result

Default Return: SimpleAnimationComponentRequests::Result::Failure

StopAllAnimations

Stops all animations that are being played on all layers.

Syntax

```cpp
Result StopAllAnimations()
```

Returns: A Result indicating whether all animations were stopped.

Return Type: SimpleAnimationComponentRequests::Result
**Default Return:** SimpleAnimationComponentRequests::Result::Failure

**StopAnimationsOnLayer**
Stops the animation currently playing on the specified layer.

**Syntax**
```
Result StopAnimationsOnLayer(AnimatedLayer::LayerId layerId, float blendOutTime)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layerId</td>
<td>AnimatedLayer::LayerId</td>
<td>Identifier for the layer that is to stop its animation (0 - AnimatedLayer::s_maxActiveAnimatedLayers)</td>
</tr>
<tr>
<td>blendOutTime</td>
<td>float</td>
<td>Time that the animations take to blend out.</td>
</tr>
</tbody>
</table>

**Returns:** A Result indicating whether the animation on the indicated layer was stopped.

**Return Type:** SimpleAnimationComponentRequests::Result

**Default Return:** SimpleAnimationComponentRequests::Result::Failure

**SetPlaybackSpeed**
Changes the playback speed for a particular layer.

**Syntax**
```
Result SetPlaybackSpeed(AnimatedLayer::LayerId layerId, float playbackSpeed)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layerId</td>
<td>AnimatedLayer::LayerId</td>
<td>Identifier for the layer whose speed should be changed.</td>
</tr>
<tr>
<td>playbackSpeed</td>
<td>float</td>
<td>The playback speed.</td>
</tr>
</tbody>
</table>

**Returns:** A Result indicating whether the animation on the indicated layer was updated or not. A failure likely indicated that no animation is playing on the specified layer.

**Return Type:** SimpleAnimationComponentRequests::Result

**Default Return:** SimpleAnimationComponentRequests::Result::Failure

**SimpleAnimationComponentNotificationBus**
This EBus interfaces handles events sent by the simple animation component.

**OnAnimationStarted**
Informs all listeners about an animation being started on a layer.

**Syntax**
void OnAnimationStarted(const AnimatedLayer& animatedLayer)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>animatedLayer</td>
<td>AnimatedLayer</td>
<td>Specifies the name and parameters of the animation that was started.</td>
</tr>
</tbody>
</table>

**OnAnimationStopped**

Informs all listeners about an animation being stopped on the indicated layer.

**Syntax**

void OnAnimationStopped(const AnimatedLayer::LayerId animatedLayer)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>animatedLayer</td>
<td>AnimatedLayer::LayerId</td>
<td>Specifies the name and parameters of the animation that was stopped.</td>
</tr>
</tbody>
</table>

**AudioEnvironmentComponentRequestBus**

This EBus interface handles messages serviced by AudioEnvironmentComponent instances. The environment refers to the effects (primarily the auxiliary effects) that the bus sends. See AudioEnvironmentComponent.cpp for details.

**SetAmount**

Sets an environment amount on the default assigned environment.

**Syntax**

void SetAmount(float amount)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>float</td>
<td>The amount for the environment.</td>
</tr>
</tbody>
</table>

**SetEnvironmentAmount**

Set an environment amount, specify an environment name at run time (that is, a script).

**Syntax**

void SetEnvironmentAmount(const char* environmentName,float amount)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentName</td>
<td>char</td>
<td>The name of the environment.</td>
</tr>
</tbody>
</table>
### Component Entity Lua API Reference

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount</td>
<td>float</td>
<td>The amount for the environment.</td>
</tr>
</tbody>
</table>

#### AudioListenerComponentRequestBus

This EBus interface handles messages serviced by `AudioListenerComponent` instances.

#### SetRotationEntity

Sets the entity for which the audio listener tracks rotation.

**Syntax**

```c
void SetRotationEntity(const AZ::EntityId entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AZ::EntityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

#### SetPositionEntity

Sets the entity for which the audio listener tracks position.

**Syntax**

```c
void SetPositionEntity(const AZ::EntityId entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AZ::EntityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

#### SetFullTransformEntity

Essentially the same as calling `SetRotationEntity` and `SetPositionEntity` on the same entity.

**Syntax**

```c
void SetFullTransformEntity(const AZ::EntityId entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AZ::EntityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

#### AudioRtpcComponentRequestBus

This EBus interface handles messages serviced by `AudioRtpcComponent` instances. RTPC stands for Real-Time Parameter Control. The `AudioRtpcComponent` is used by the game to configure parameters in the audio engine. See `AudioRtpcComponent.cpp` for details.
**SetValue**

Sets an RTPC value for the RTPC name that has been serialized with the component.

**Syntax**

```c
void SetValue(float value)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>float</td>
<td>The RTPC value to set.</td>
</tr>
</tbody>
</table>

**SetRtpcValue**

Use to manually specify an RTPC name and value at run time for use in scripting.

**Syntax**

```c
void SetRtpcValue(const char* rtpcName, float value)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtpcName</td>
<td>char</td>
<td>Specifies an RTPC name to use.</td>
</tr>
<tr>
<td>value</td>
<td>float</td>
<td>Specifies a value for the RTPC name supplied.</td>
</tr>
</tbody>
</table>

**AudioSwitchComponentRequestBus**

This EBus interface handles messages serviced by AudioSwitchComponent instances. A Switch is an object that can be in one State at a time, but whose State value can be changed at run time. For example, a Switch called SurfaceMaterial might have states such as 'Grass', 'Snow', 'Metal', or 'Wood'. See AudioSwitchComponent.h for details.

**SetState**

Sets the name of the state on the default assigned switch.

**Syntax**

```c
void SetState(const char* stateName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateName</td>
<td>char</td>
<td>Specifies the name of the state to set.</td>
</tr>
</tbody>
</table>

**SetSwitchState**

Sets the specified switch to the specified state.

**Syntax**

```c
```
```c
void SetSwitchState(const char* switchName, const char* stateName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switchName</td>
<td>char</td>
<td>The name of the switch to set.</td>
</tr>
<tr>
<td>stateName</td>
<td>char</td>
<td>The name of the state to set on the specified switch.</td>
</tr>
</tbody>
</table>

**AudioTriggerComponentRequestBus**

This EBus interface handles messages serviced by AudioTriggerComponent instances. You can use the AudioTriggerComponent to execute, stop, and control ATL triggers. You can serialize the name of the trigger with the component or manually specify the name at run time for use in scripting. Only one AudioTriggerComponent is allowed on an entity, but the interface supports firing multiple ATL triggers.

**Play**

Executes the play trigger if the play trigger is set.

**Syntax**

```c
void Play()
```

**Stop**

Executes the stop trigger if one is set; otherwise, stops the play trigger.

**Syntax**

```c
void Stop()
```

**ExecuteTrigger**

Executes the specified ATL trigger.

**Syntax**

```c
void ExecuteTrigger(const char* triggerName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggerName</td>
<td>char</td>
<td>Specifies the name of the trigger to execute.</td>
</tr>
</tbody>
</table>

**KillTrigger**

Kills the specified ATL Trigger.

**Syntax**

```c
void KillTrigger(const char* triggerName)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggerName</td>
<td>char</td>
<td>Specifies the name of the trigger to remove.</td>
</tr>
</tbody>
</table>

**KillAllTriggers**

Forces a removal of triggers that are active on the underlying proxy.

**Syntax**

```cpp
void KillAllTriggers()
```

**SetMovesWithEntity**

Specifies whether the trigger should be repositioned as the entity moves.

**Syntax**

```cpp
void SetMovesWithEntity(bool shouldTrackEntity)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shouldTrackEntity</td>
<td>bool</td>
<td>Specify true to have the trigger track the entity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify false to have the trigger not track the entity.</td>
</tr>
</tbody>
</table>

**AudioTriggerComponentNotificationBus**

This EBus interface handles messages sent by AudioTriggerComponent instances.

**OnTriggerFinished**

Notifies when a trigger instance has finished.

**Syntax**

```cpp
void OnTriggerFinished(const Audio::TAudioControlID triggerID)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggerID</td>
<td>Audio::TAudioControlID</td>
<td>The ID of the trigger.</td>
</tr>
</tbody>
</table>

**FloatGameplayNotificationBus (AZ::GameplayNotificationBus<float>)**

This version of the GameplayNotificationBus EBus interface handles float-based gameplay notifications.

**OnGameplayEventAction**

Event sent when the specified GameplayEventAction has occurred.
OnGameplayEventFailed
Event sent when the given GameplayEventAction has failed.

Vector3GameplayNotificationBus
This version of the GameplayNotificationBus EBus interface handles Vector3-based gameplay notifications.

OnGameplayEventAction
Event sent when the given GameplayEventAction has occurred.

OnGameplayEventFailed
Event sent when the given GameplayEventAction has failed.

StringGameplayNotificationBus
(AZ::GameplayNotificationBus<const AZStd::wstring>)
This version of the GameplayNotificationBus EBus interface handles string-based gameplay notifications.

OnGameplayEventAction
Event sent when the given GameplayEventAction has occurred.

OnGameplayEventFailed
Event sent when the given GameplayEventAction has failed.

EntityIdGameplayNotificationBus
(AZ::GameplayNotificationBus<AZ::EntityId>)
This EBus interface handles EntityId-based gameplay notifications. It is a specialization of the GameplayNotificationBus.

OnGameplayEventAction
Event sent when the given GameplayEventAction has occurred.

OnGameplayEventFailed
Event sent when the given GameplayEventAction has failed.

CryCharacterPhysicsRequestBus
This EBus interface handles messages serviced by Cry character physics.

Move
Requests movement from Living Entity.

Syntax
void Move(const AZ::Vector3& velocity, int jump)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity</td>
<td>AZ::Vector3</td>
<td>Requested velocity (direction and magnitude).</td>
</tr>
<tr>
<td>jump</td>
<td>int</td>
<td>Controls how the value for the velocity parameter is applied within a Living Entity. To change the velocity to the new value, specify 1. To add the value to the current velocity, specify 2.</td>
</tr>
</tbody>
</table>

**ConstraintComponentRequestBus**

This EBus interface handles messages serviced by instances of the Constraint component. A Constraint component facilitates the creation of a physics constraint between two entities or an entity and a point in the world. Both entities must have a component that provides the physics service.

**SetConstraintEntities**

Sets the entity that owns the constraint and the target of the constraint.

**Syntax**

```cpp
void SetConstraintEntities(const AZ::EntityId& owningEntity, const AZ::EntityId& targetEntity)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owningEntity</td>
<td>AZ::EntityId</td>
<td>Specifies the ID of the entity that owns the constraint.</td>
</tr>
<tr>
<td>targetEntity</td>
<td>AZ::EntityId</td>
<td>Specifies the ID of the entity that is the target of the constraint. The target is invalid if constrained to world space.</td>
</tr>
</tbody>
</table>

**SetConstraintEntitiesWithPartIds**

Sets the entity that owns the constraint, the target entity, and the animation part IDs (bone IDs) for the constraint to be attached to.

**Syntax**

```cpp
void SetConstraintEntitiesWithPartIds(const AZ::EntityId& owningEntity, int ownerPartId, const AZ::EntityId& targetEntity, int targetPartId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owningEntity</td>
<td>AZ::EntityId</td>
<td>Specifies the ID of the entity that owns the constraint.</td>
</tr>
<tr>
<td>ownerPartId</td>
<td>int</td>
<td>Specifies the ID of the owner part (the bone ID) for the constraint.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>targetEntity</td>
<td>AZ::EntityId</td>
<td>Specifies the ID of the entity that is the target of the constraint.</td>
</tr>
<tr>
<td>targetPartId</td>
<td>int</td>
<td>Specifies the ID of the target part (the bone ID) for the constraint.</td>
</tr>
</tbody>
</table>

**EnableConstraint**

Enable all constraints on the current entity.

**Syntax**

```plaintext
void EnableConstraint()
```

**DisableConstraint**

Disable all constraints on the current entity.

**Syntax**

```plaintext
void DisableConstraint()
```

**ConstraintComponentNotificationBus**

This EBus interface handles messages dispatched by the Constraint component.

**OnConstraintEntitiesChanged**

This event fires when either the constraint owner or target changes. The target is invalid if constrained to world space.

**Note**

This event also fires when `partId` values change.

**Syntax**

```plaintext
void OnConstraintEntitiesChanged(const AZ::EntityId& oldOwner, const AZ::EntityId& oldTarget, const AZ::EntityId& newOwner, const AZ::EntityId& newTarget)
```
OnConstraintEnabled
Fires when constraints have been enabled on the current entity.

Syntax

void OnConstraintEnabled()

OnConstraintDisabled
Fires when a constraint has been disabled.

Syntax

void OnConstraintDisabled()

PhysicsComponentRequestBus
This EBus interface handles messages serviced by the in-game Physics component.

EnablePhysics
Makes the entity a participant in the physics simulation.

Syntax

void EnablePhysics()

DisablePhysics
Stops the entity from participating in the physics simulation

Syntax

void DisablePhysics()

IsPhysicsEnabled
Checks if physics are enabled on the current entity.

Syntax

bool IsPhysicsEnabled()

Returns: true if physics are enabled; false otherwise.

Return Type: bool

Default Return: false

AddImpulse
Applies the specified impulse to the entity.

Syntax
void AddImpulse(const AZ::Vector3& impulse)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>impulse</td>
<td>AZ::Vector3</td>
<td>Vector of the impulse.</td>
</tr>
</tbody>
</table>

**AddAngularImpulse**

Applies an angular impulse to the entity.

**Syntax**

void AddAngularImpulse(const AZ::Vector3& /*impulse*/, const AZ::Vector3& worldSpacePivot)

| Parameter  | Type                              | Description                                                         |
|------------|-----------------------------------|                                                                    |
| impulse    | AZ::Vector3                        | Vector of the impulse.                                              |
| worldSpacePivot | AZ::Vector3               | Vector of the world space pivot to apply to the entity.             |

**GetVelocity**

Retrieves the velocity of the entity.

**Syntax**

AZ::Vector3 GetVelocity()

**Returns:** The velocity of the entity.

**Return Type:** AZ::Vector3

**Default Return:** AZ::Vector3::CreateZero()

**SetVelocity**

Sets the velocity of the entity.

**Syntax**

void SetVelocity(const AZ::Vector3& velocity)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>velocity</td>
<td>AZ::Vector3</td>
<td>Specifies the velocity to set.</td>
</tr>
</tbody>
</table>

**GetAcceleration**

Gets the linear acceleration of the entity.
Syntax

```python
AZ::Vector3 GetAcceleration()
```

**Returns:** A vector containing the linear acceleration of the entity.

**Return Type:** AZ::Vector3

**Default Return:** AZ::Vector3::CreateZero()

**GetAngularVelocity**

Gets the angular velocity of the entity.

**Syntax**

```python
AZ::Vector3 GetAngularVelocity()
```

**Returns:** A vector containing the angular velocity of the entity.

**Return Type:** AZ::Vector3

**Default Return:** AZ::Vector3::CreateZero()

**SetAngularVelocity**

Sets the angular velocity of the entity to the specified amount.

**Syntax**

```python
void SetAngularVelocity(const AZ::Vector3& angularVelocity)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>angularVelocity</td>
<td>AZ::Vector3</td>
<td>The angular velocity to set.</td>
</tr>
</tbody>
</table>

**GetAngularAcceleration**

Gets the angular acceleration of the entity

**Syntax**

```python
AZ::Vector3 GetAngularAcceleration()
```

**Returns:** A vector containing the angular acceleration of the entity.

**Return Type:** AZ::Vector3

**Default Return:** AZ::Vector3::CreateZero()

**GetMass**

Retrieves the mass of the entity.
Syntax

float GetMass()

Returns: The mass of the entity.
Return Type: float
Default Return: 0.0f

PhysicsComponentNotificationBus

This bus handles events emitted by a Physics component and by the Physics system.

OnPhysicsEnabled

Fires when an entity begins participating in the physics simulation. If the entity is active when a handler connects to the bus, then OnPhysicsEnabled() is immediately dispatched.

Note
If physics is enabled, OnPhysicsEnabled fires immediately upon connecting to the bus.

Syntax

void OnPhysicsEnabled()

OnPhysicsDisabled

Fires when an entity ends its participation in the physics simulation.

Syntax

void OnPhysicsDisabled()

OnCollision

Fires when an entity collides with another entity.

Syntax

void OnCollision(const Collision& collision)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collision</td>
<td>Collision</td>
<td>Contains information about the collision that occurred. See the following Collision struct.</td>
</tr>
</tbody>
</table>

struct Collision
{
    AZ_TYPE_INFO(Collision, "{33756BD4-24D4-4DAE-A849-537114D52F7D}");
    AZ_CLASS_ALLOCATOR(Collision, AZ::SystemAllocator, 0);
    AZ::EntityId m_entity; // ID of other entity involved in event
```csharp
AZ::Vector3 m_position;       // Contact point in world coordinates
AZ::Vector3 m_normal;         // Normal to the collision
float m_impulse;              // Impulse applied by the collision resolver
AZ::Vector3 m_velocityA;      // Velocities of the first entity involved in the collision
AZ::Vector3 m_velocityB;      // Velocities of the second entity involved in the collision
float m_massA;                // Masses of the first entity involved in the collision
float m_massB;                // Masses of the second entity involved in the collision
}

PhysicsSystemRequestBus

Requests for the physics system

RayCast

Casts a ray and retrieves a list of results.

Syntax

RayCastHit RayCast(const AZ::Vector3& begin, const AZ::Vector3& direction, float maxDistance, AZ::u32 maxHits, AZ::u32 query)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td>const AZ::Vector3&amp;</td>
<td>The origin of the ray</td>
</tr>
<tr>
<td>direction</td>
<td>const AZ::Vector3&amp;</td>
<td>The direction for the ray to travel</td>
</tr>
<tr>
<td>maxDistance</td>
<td>float</td>
<td>The maximum distance the ray will travel</td>
</tr>
<tr>
<td>maxHits</td>
<td>AZ::u32</td>
<td>The maximum number of hits to return from the search.</td>
</tr>
<tr>
<td>query</td>
<td>AZ::u32</td>
<td>The entity types to hit. See the PhysicalEntityTypes enum that follows.</td>
</tr>
</tbody>
</table>

Returns: A RayCastHit struct. For details, see the code listing that follows.

Return Type: PhysicsSystemRequests::RayCastHit

Default Return: RayCastHit()

struct RayCastHit
{
    AZ_TYPE_INFO(RayCastHit, "{3D8FA68C-A145-44B4-BA18-F3405D83A9DF}");
    AZ_CLASS_ALLOCATOR(RayCastHit, AZ::SystemAllocator, 0);

    float m_distance = 0.0f;       // The distance from RayCast begin to the hit.
    AZ::Vector3 m_position;        // The position of the hit in world space.
    AZ::Vector3 m_normal;          // The normal of the surface hit.
    AZ::EntityId m_entityId;       // The ID of the AZ::Entity hit, or
    // AZ::InvalidEntityId if hit object is not an AZ::Entity.
};
```
**RagdollPhysicsRequestBus**

Messages serviced by the Cry character physics ragdoll behavior.

**EnterRagdoll**

Causes an entity with a skinned mesh component to disable its current physics and enable ragdoll physics.

**Syntax**

```plaintext
void EnterRagdoll()
```

**ExitRagdoll**

Causes the ragdoll component to deactivate itself and reenable the entity's physics component.

**Syntax**

```plaintext
void ExitRagdoll()
```

**DecalComponentRequestBus**

This EBus interface handles messages serviced by the Decal component.

**SetVisibility**

Specifies the decal's visibility.

**Syntax**

```plaintext
void SetVisibility(bool visible)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>bool</td>
<td>Specify true to make the decal visible, false to hide it.</td>
</tr>
</tbody>
</table>

**Show**

Makes the decal visible.

**Syntax**

```plaintext
void Show()
```

**Hide**

Hides the decal.

**Syntax**

```plaintext
void Hide()
```
**LensFlareComponentRequestBus**

This EBus interface handles messages serviced by the Lens Flare component.

**SetLensFlareState**

Controls the lens flare state.

**Syntax**

```lua
void SetLensFlareState(State state)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>State</td>
<td>Specify <code>on</code> to turn on the lens flare; specify <code>off</code> to turn it off.</td>
</tr>
</tbody>
</table>

**TurnOnLensFlare**

Turns the lens flare on.

**Syntax**

```lua
void TurnOnLensFlare()
```

**TurnOffLensFlare**

Turns the lens flare off.

**Syntax**

```lua
void TurnOffLensFlare()
```

**ToggleLensFlare**

Toggles the lens flare state.

**Syntax**

```lua
void ToggleLensFlare()
```

**LensFlareComponentNotificationBus**

This EBus interface handles events dispatched by the Lens Flare component.

**LensFlareTurnedOn**

Notifies that the lens flare has been turned on.

**Syntax**

```lua
void LensFlareTurnedOn()
```
**LensFlareTurnedOff**

Notifies that the lens flare has been turned off.

**Syntax**

```lua
void LensFlareTurnedOff()
```

**LightComponentRequestBus**

This EBus interfaces handles messages serviced by the light component.

**SetLightState**

Controls the light state.

**Syntax**

```lua
void SetLightState(State state)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>State</td>
<td>Specify <strong>On</strong> to turn on the light; specify <strong>Off</strong> to turn it off.</td>
</tr>
</tbody>
</table>

**TurnOnLight**

Turns the light on.

**Syntax**

```lua
void TurnOnLight()
```

**TurnOffLight**

Turns the light off.

**Syntax**

```lua
void TurnOffLight()
```

**ToggleLight**

Toggles the light state.

**Syntax**

```lua
void ToggleLight()
```

**LightComponentNotificationBus**

Light component notifications.
LightTurnedOn

Event sent when a light component is turned on.

Syntax

```lua
void LightTurnedOn()
```

LightTurnedOff

Event sent when a light component is turned off.

Syntax

```lua
void LightTurnedOff()
```

ParticleComponentRequestBus

Provides access to the particle component.

SetVisibility

Specifies the visibility of the particle component.

Syntax

```lua
void SetVisibility(bool visible)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>bool</td>
<td>Specify true to make the particle component visible; false to hide it.</td>
</tr>
</tbody>
</table>

Show

Makes the particle component visible.

Syntax

```lua
void Show()
```

Hide

Hides the particle component.

Syntax

```lua
void Hide()
```

SetupEmitter

Sets up an effect emitter with the specified name and settings.
Syntax

void SetupEmitter(const AZStd::string& emitterName, const ParticleEmitterSettings& settings)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emitterName</td>
<td>const AZStd::string&amp;</td>
<td>The name of the emitter to set up.</td>
</tr>
<tr>
<td>settings</td>
<td>const ParticleEmitterSettings&amp;</td>
<td>Contains particle emitter settings. For more information, see ParticleComponent.cpp.</td>
</tr>
</tbody>
</table>

**SimpleStateComponentRequestBus**

This EBus interface handles messages serviced by the Simple State component. The Simple State component provides a simple state machine. Each state is represented by a name and zero or more entities that are activated when the state is entered and deactivated when the state is left.

**SetState**

Sets the active state

**Syntax**

```cpp
void SetState(const char* stateName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateName</td>
<td>char</td>
<td>The name of the state.</td>
</tr>
</tbody>
</table>

**SetStateByIndex**

Sets the active state using a 0-based index.

**Syntax**

```cpp
void SetStateByIndex(AZ::u32 stateIndex)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateIndex</td>
<td>AZ::u32</td>
<td>The 0-based index of the state.</td>
</tr>
</tbody>
</table>

**SetToNextState**

Advances to the next state. If the next state is null, the first state is set.

**Syntax**

```cpp
void SetToNextState()
```
**SetToPreviousState**
Sets the previous state. If the previous state is null, the end state is set.

**Syntax**

```c
void SetToPreviousState()
```

**SetToFirstState**
Sets the first state.

**Syntax**

```c
void SetToFirstState()
```

**SetToLastState**
Sets the last state.

**Syntax**

```c
void SetToLastState()
```

**GetNumStates**
Get the number of states.

**Syntax**

```c
AZ::u32 GetNumStates()
```

**Returns:** The number of states.

**Return Type:** AZ::u32

**Default Return:** 0

**GetCurrentState**
Gets the current state.

**Syntax**

```c
const char* GetCurrentState()
```

**Returns:** The current state.

**Return Type:** const char*

**Default Return:** nullptr

**SimpleStateComponentNotificationBus**

This EBus interface handles events dispatched by the Simple State component.
**OnStateChanged**

Notify that the state has changed from `oldState` to `newState`.

**Syntax**

```c
void OnStateChanged(const char* oldState, const char* newState)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldState</td>
<td>char</td>
<td>The name of the old state.</td>
</tr>
<tr>
<td>newState</td>
<td>char</td>
<td>The name of the new state.</td>
</tr>
</tbody>
</table>

**SpawnerComponentRequestBus**

This EBus interface handles messages serviced by the `SpawnerComponent`.

**Spawn**

Spawns the selected slice at the entity's location.

**Syntax**

```c
AzFramework::SliceInstantiationTicket Spawn()
```

**Returns:** A slice instantiation ticket.

**Return Type:** `AzFramework::SliceInstantiationTicket`

**Default Return:** `AzFramework::SliceInstantiationTicket()`

**SpawnRelative**

Spawns the selected slice at the entity's location with the specified relative offset.

**Syntax**

```c
AzFramework::SliceInstantiationTicket SpawnRelative(const AZ::Transform& relative)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relative</td>
<td>AZ::Transform</td>
<td>Relative offset from the entity's location.</td>
</tr>
</tbody>
</table>

**Returns:** A slice instantiation ticket.

**Return Type:** `AzFramework::SliceInstantiationTicket`

**Default Return:** `AzFramework::SliceInstantiationTicket()`

**SpawnAbsolute**

Spawns the selected slice at the specified world transform.

**Syntax**
AzFramework::SliceInstantiationTicket SpawnAbsolute(const AZ::Transform& world)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>world</td>
<td>const AZ::Transform&amp;</td>
<td>Specifies the world transform at which to spawn the selected slice.</td>
</tr>
</tbody>
</table>

**Returns:** A slice instantiation ticket.

**Return Type:** AzFramework::SliceInstantiationTicket

**Default Return:** AzFramework::SliceInstantiationTicket()

### SpawnerComponentNotificationBus

This EBus interface handles events dispatched by the SpawnerComponent.

**OnSpawnBegin**

Notifies that a slice has been spawned, but that its entities have not yet been activated. OnEntitySpawned events are about to be dispatched.

**Syntax**

```cpp
void OnSpawnBegin(const AzFramework::SliceInstantiationTicket& ticket)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ticket</td>
<td>AzFramework::SliceInstantiationTicket</td>
<td>The slice instantiation ticket.</td>
</tr>
</tbody>
</table>

**OnSpawnEnd**

Notifies that a spawn has been completed. All OnEntitySpawned events have been dispatched.

**Syntax**

```cpp
void OnSpawnEnd(const AzFramework::SliceInstantiationTicket& ticket)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ticket</td>
<td>AzFramework::SliceInstantiationTicket</td>
<td>The slice instantiation ticket.</td>
</tr>
</tbody>
</table>

**OnEntitySpawned**

Notifies that an entity has spawned. This event is called once for each entity spawned in a slice.

**Syntax**

```cpp
void OnEntitySpawned(const AzFramework::SliceInstantiationTicket& ticket, const AZ::EntityId& spawnedEntities)
```
**Component Entity Lua API Reference**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ticket</td>
<td>AzFramework::SliceInstantiationTicket</td>
<td>The slice instantiation ticket.</td>
</tr>
<tr>
<td>spawnedEntityId</td>
<td>AZ::EntityId</td>
<td>The ID of the spawned entity.</td>
</tr>
</tbody>
</table>

**TagComponentRequestBus**

Provides services for managing tags on entities.

**HasTag**

Checks for a specified tag on an entity.

**Syntax**

```c
bool HasTag(const Tag&)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>Tag</td>
<td>The tag to query for.</td>
</tr>
</tbody>
</table>

**Returns**: true if the entity has the specified tag; false otherwise.

**Return Type**: bool

**Default Return**: false

**AddTag**

Adds the specified tag to the entity if it doesn't already have it.

**Syntax**

```c
void AddTag(const Tag&)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Tag</td>
<td>The tag to add.</td>
</tr>
</tbody>
</table>

**AddTags**

Adds a specified list of tags to the entity if the list does not exist on the entity.

**Syntax**

```c
void AddTags(const Tags& tags)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tags</td>
<td>Tags</td>
<td>The list of tags to add.</td>
</tr>
</tbody>
</table>
RemoveTag

Removes a specified tag from the entity if the tag is present.

Syntax

```c
void RemoveTag(const Tag& tag);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>Tag</td>
<td>The tag to remove.</td>
</tr>
</tbody>
</table>

RemoveTags

Removes the specified list of tags from the entity if the list exists on the entity.

Syntax

```c
void RemoveTags(const Tags& tags);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tags</td>
<td>Tags</td>
<td>The list of tags to remove.</td>
</tr>
</tbody>
</table>

GetTags

Retrieves the list of tags on the entity.

Syntax

```c
const Tags& GetTags();
```

Returns: A list of the tags on the entity.

Return Type: static Tags

Default Return: s_emptyTags

TagGlobalRequestBus

Provides services for querying Tags on entities.

RequestTaggedEntities

Queries for tagged entities. Handlers respond if they have the tag (that is, they are listening on the tag’s channel). Use AZ::EBusAggregateResults to handle more than the first responder.

Syntax

```c
const AZ::EntityId RequestTaggedEntities();
```

Returns: The ID of an entity that has a tag.

Return Type: const AZ::EntityId
Default Return: s_invalidEntityId

TagGlobalNotificationBus

Handler for global Tag component notifications.

OnEntityTagAdded

Notifies that a tag has been added to an entity. When connecting to the tag global notification bus, your OnEntityTagAdded handler fires once for each entity that already has a tag. After the initial connection, you are alerted whenever a new entity gains or loses a tag.

Syntax

void OnEntityTagAdded(const AZ::EntityId&)

OnEntityTagRemoved

Notifies that a Tag has been removed from an entity.

Syntax

void OnEntityTagRemoved(const AZ::EntityId&)

TagComponentNotificationsBus

Provides notifications regarding tags on entities.

OnTagAdded

Notifies listeners when a tag has been added.

Syntax

void OnTagAdded(const Tag&)

OnTagRemoved

Notifies listeners when a tag is removed.

Syntax

void OnTagRemoved(const Tag&)

TriggerAreaRequestsBus

This EBus interface services requests made to the Trigger Area component.

AddRequiredTag

Adds a required tag to the activation filtering criteria of the current component.

Syntax

void AddRequiredTag(const Tag& requiredTag)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requiredTag</td>
<td>Tag</td>
<td>The tag to add to the activation filtering criteria.</td>
</tr>
</tbody>
</table>

**RemoveRequiredTag**

Removes a required tag from the activation filtering criteria of the current component.

**Syntax**

```cpp
void RemoveRequiredTag(const Tag& requiredTag)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requiredTag</td>
<td>Tag</td>
<td>The tag to remove from the activation filtering criteria.</td>
</tr>
</tbody>
</table>

**AddExcludedTag**

Adds an excluded tag to the activation filtering criteria of the current component.

**Syntax**

```cpp
void AddExcludedTag(const Tag& excludedTag)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>excludedTag</td>
<td>Tag</td>
<td>The excluded tag to add to the activation filtering criteria.</td>
</tr>
</tbody>
</table>

**RemoveExcludedTag**

Removes an excluded tag from the activation filtering criteria of the current component.

**Syntax**

```cpp
void RemoveExcludedTag(const Tag& excludedTag)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>excludedTag</td>
<td>Tag</td>
<td>The excluded tag to remove from the activation filtering criteria.</td>
</tr>
</tbody>
</table>

**TriggerAreaNotificationBus**

This EBus handles events for a given trigger area when an entity enters or leaves.

**OnTriggerAreaEntered**

Notifies when an entity enters the trigger area.
Syntax

```c
void OnTriggerAreaEntered(AZ::EntityId enteringEntityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enteringEntityId</td>
<td>AZ::EntityId</td>
<td>The ID of the entity that entered the trigger area.</td>
</tr>
</tbody>
</table>

**OnTriggerAreaExited**

Notifies when an entity exits the trigger area.

Syntax

```c
void OnTriggerAreaExited(AZ::EntityId exitingEntityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitingEntityId</td>
<td>AZ::EntityId</td>
<td>The ID of the entity that exited the trigger area.</td>
</tr>
</tbody>
</table>

**TriggerAreaEntityNotificationBus**

Events fired for a specified trigger when the trigger area has been entered or exited.

**OnEntityEnteredTriggerArea**

Notifies when an `enteringEntityId` instance has entered the specified trigger area.

Syntax

```c
void OnEntityEnteredTriggerArea(AZ::EntityId triggerId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggerId</td>
<td>AZ::EntityId</td>
<td>The ID of the trigger that has been entered.</td>
</tr>
</tbody>
</table>

**OnEntityExitedTriggerArea**

Notifies when an `enteringEntityId` instance has exited the specified trigger area.

Syntax

```c
void OnEntityExitedTriggerArea(AZ::EntityId triggerId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggerId</td>
<td>AZ::EntityId</td>
<td>The ID of the trigger that has been exited.</td>
</tr>
</tbody>
</table>
BoxShapeComponentRequestsBus

Services provided by the Box Shape component.

GetBoxConfiguration

Retrieves the box configuration.

Syntax

BoxShapeConfiguration GetBoxConfiguration()

Return Type: BoxShapeConfiguration
Default Return: BoxShapeConfiguration()

SetBoxDimensions

Sets new dimensions for the Box Shape.

Syntax

void SetBoxDimensions(AZ::Vector3 newDimensions)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newDimensions</td>
<td>AZ::Vector3</td>
<td>Specifies dimensions along the X, Y, and Z axes.</td>
</tr>
</tbody>
</table>

CapsuleShapeComponentRequestsBus

Services provided by the Capsule Shape Component.

GetCapsuleConfiguration

Retrieves the capsule configuration.

Syntax

CapsuleShapeConfiguration GetCapsuleConfiguration()

Returns: The capsule configuration.
Return Type: CapsuleShapeConfiguration
Default Return: CapsuleShapeConfiguration()

SetHeight

Sets the end to end height of capsule, including the cylinder and both caps.

Syntax

void SetHeight(float newHeight)
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newHeight</td>
<td>float</td>
<td>Specifies the new height of the capsule.</td>
</tr>
</tbody>
</table>

### SetRadius

Sets the radius of the capsule.

**Syntax**

```c
void SetRadius(float newRadius)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newRadius</td>
<td>float</td>
<td>Specifies the new radius of the capsule.</td>
</tr>
</tbody>
</table>

### CylinderShapeComponentRequestsBus

This EBus interface handles messages for the Cylinder Shape component.

#### GetCylinderConfiguration

Retrieves the cylinder configuration.

**Syntax**

```c
CylinderShapeConfiguration GetCylinderConfiguration()
```

**Returns:** The cylinder configuration.

**Return Type:** CylinderShapeConfiguration

**Default Return:** CylinderShapeConfiguration()

#### SetHeight

Sets the height of the cylinder.

**Syntax**

```c
void SetHeight(float newHeight)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newHeight</td>
<td>float</td>
<td>Specifies the height of the cylinder.</td>
</tr>
</tbody>
</table>

### SetRadius

Sets the radius of the cylinder.
Syntax

```c
void SetRadius(float newRadius)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newRadius</td>
<td>float</td>
<td>Specifies the radius of the cylinder.</td>
</tr>
</tbody>
</table>

ShapeComponentRequestsBus

Handles requests for services provided by the Shape component.

**GetShapeType**

Retrieves the type of shape of a component.

**Syntax**

```c
AZ::Crc32 GetShapeType()
```

**Returns:** A Crc32 value that indicates the type of shape of the current component.

**Return Type:** AZ::Crc32

**Default Return:** AZ::Crc32()

**IsPointInside**

Checks if a given point is inside or outside a shape.

**Syntax**

```c
bool IsPointInside(const AZ::Vector3& point)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>AZ::Vector3</td>
<td>Specifies the coordinates of the point to be tested.</td>
</tr>
</tbody>
</table>

**Returns:** A bool value that indicates whether the point is inside or out.

**Return Type:** bool

**Default Return:** false

**DistanceFromPoint**

Retrieves the minimum distance the specified point is from the shape.

**Syntax**

```c
float DistanceFromPoint(const AZ::Vector3& point)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>AZ::Vector3</td>
<td>Specifies the coordinates of the point from which to calculate distance.</td>
</tr>
</tbody>
</table>

**Returns:** A float that indicates the distance the point is from the shape.

**Return Type:** float

**Default Return:** 0.f

### DistanceSquaredFromPoint

Retrieves the minimum squared distance the specified point is from the shape.

**Syntax**

```lua
float DistanceSquaredFromPoint(const AZ::Vector3& point)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>AZ::Vector3</td>
<td>Specifies the coordinates of the point from which to calculate the squared distance.</td>
</tr>
</tbody>
</table>

**Returns:** A float that contains the minimum squared distance the specified point is from the shape.

**Return Type:** float

**Default Return:** 0.f

### ShapeComponentNotificationsBus

Notifications sent by the shape component.

**OnShapeChanged**

Notifies that the shape component has been modified.

**Syntax**

```lua
void OnShapeChanged(ShapeChangeReasons changeReason)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>changeReason</td>
<td>ShapeChangeReasons</td>
<td>Informs listeners of the reason for this shape change (transform change, the shape dimensions being altered.)</td>
</tr>
</tbody>
</table>

### SphereShapeComponentRequestsBus

Services provided by the Sphere Shape Component
GetSphereConfiguration
Retrieve the sphere configuration.

Syntax
```
SphereShapeConfiguration GetSphereConfiguration()
```

**Returns:** The sphere configuration.

**ReturnType:** SphereShapeConfiguration

**Default Return:** SphereShapeConfiguration()

SetRadius
Sets the specified radius for the sphere shape component.

Syntax
```
void SetRadius(float newRadius)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newRadius</td>
<td>float</td>
<td>Specifies the radius of the sphere shape.</td>
</tr>
</tbody>
</table>

EntityBus
Dispatches events specific to a given entity.

OnEntityActivated
Notifies when entity activation has completed. If the entity is active when a handler connects to the bus, then the OnEntityActivated event is sent immediately.

Syntax
```
void OnEntityActivated(const AZ::EntityId&)
```

OnEntityDeactivated
Notifies when the entity is about to be deactivated.

Syntax
```
void OnEntityDeactivated(const AZ::EntityId&)
```

TickBus
Tick events are executed on the main game or component thread.

**Note**
Warning: Adding mutex to the tick bus degrades performance in most cases.
OnTick

Notifies the delta time if the delta from the previous tick (in seconds) and time point is its absolute value.

**Syntax**

```lua
void OnTick(float deltaTime, ScriptTimePoint time)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deltaTime</td>
<td>float</td>
<td>The latest time between ticks.</td>
</tr>
<tr>
<td>time</td>
<td>ScriptTimePoint</td>
<td>The time at the current tick.</td>
</tr>
</tbody>
</table>

TickRequestBus

Make requests from this bus to get the frame time or return the current time as seconds.

GetTickDeltaTime

Gets the latest time between ticks.

**Syntax**

```lua
float GetTickDeltaTime()
```

**Returns:** The latest time between ticks.

**Return Type:** float

**Default Return:** 0.f

GetTimeAtCurrentTick

Gets the time in seconds at the current tick.

**Syntax**

```lua
ScriptTimePoint GetTimeAtCurrentTick()
```

**Returns:** The time in seconds at the current tick.

**Return Type:** ScriptTimePoint

**Default Return:** ScriptTimePoint()

TransformNotificationBus

This EBus is a listener for transform changes.

OnTransformChanged

Notifies when the local transform of the entity has changed. A local transform update always implies a world transform change.
Syntax

```cpp
void OnTransformChanged(const Transform& local, const Transform& world)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>Transform</td>
<td>The local transform of the entity.</td>
</tr>
<tr>
<td>world</td>
<td>Transform</td>
<td>The world transform.</td>
</tr>
</tbody>
</table>

**OnParentChanged**

Notifies when the parent of an entity has changed. When the old or new parent is invalid, the invalid EntityId is equal to InvalidEntityId.

**Syntax**

```cpp
void OnParentChanged(EntityId oldParent, EntityId newParent)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldParent</td>
<td>EntityId</td>
<td>The entity ID of the old parent.</td>
</tr>
<tr>
<td>newParent</td>
<td>EntityId</td>
<td>The entity ID of the new parent.</td>
</tr>
</tbody>
</table>

**GameEntityContextRequestBus**

This EBus interfaces makes requests to the game entity context component.

**DestroyGameEntity**

Destroys an entity. The entity is deactivated immediately and is destroyed in the next tick.

**Syntax**

```cpp
void DestroyGameEntity(const AZ::EntityId& id)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>AZ::EntityId</td>
<td>The ID of the entity to be destroyed.</td>
</tr>
</tbody>
</table>

**DestroyGameEntityAndDescendants**

Destroys an entity and all its descendants, the entity and its descendants are deactivated immediately and will be destroyed the next tick.

**Syntax**

```cpp
void DestroyGameEntityAndDescendants(AZ::EntityId& id)
```
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>AZ::EntityId</td>
<td>The ID of the entity to be destroyed. The entity's descendants will also be destroyed.</td>
</tr>
</tbody>
</table>

### ActivateGameEntity

Activates an entity by the specified ID.

**Syntax**

```c++
void ActivateGameEntity(AZ::EntityId& id)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>AZ::EntityId</td>
<td>The ID of the entity to activate.</td>
</tr>
</tbody>
</table>

### DeactivateGameEntity

Deactivates an entity by the specified ID.

**Syntax**

```c++
void DeactivateGameEntity(AZ::EntityId& id)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>AZ::EntityId</td>
<td>The ID of the entity to deactivate.</td>
</tr>
</tbody>
</table>

### DestroySliceByEntity

Destroys the slice instance that contains the entity with the specified ID.

**Syntax**

```c++
bool DestroySliceByEntity(AZ::EntityId& id)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>AZ::EntityId</td>
<td>The ID of the entity to destroy.</td>
</tr>
</tbody>
</table>

**Returns**: true if the slice instance was successfully destroyed.

**Return Type**: bool

**Default Return**: false

### RandomManagerBus

Provides functions for random numbers.

**Version**: 1.28

**2752**
**RandomFloat**

Generates a random float value.

**Syntax**

```lua
float RandomFloat()
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>AZStd::string</td>
<td>The tag.</td>
</tr>
</tbody>
</table>

**Returns:** A random value between [0.0f, 1.0f).

**ReturnType:** float

**Default Return:** 0.0f

**RandomBool**

Generates a random Boolean value.

**Syntax**

```lua
bool RandomBool(const AZStd::string& tag)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>AZStd::string</td>
<td>The tag.</td>
</tr>
</tbody>
</table>

**Returns:** A random Boolean value.

**ReturnType:** bool

**Default Return:** false

**RandomInt**

Generates a random unsigned integer value.

**Syntax**

```lua
unsigned int RandomInt(const AZStd::string& tag)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>AZStd::string</td>
<td>The tag.</td>
</tr>
</tbody>
</table>

**Returns:** A random unsigned integer value.

**ReturnType:** unsigned int

**Default Return:** 0
**RandomInRange**

Generates a random unsigned integer value within a specified range.

**Syntax**

```cpp
unsigned int RandomInRange(const AZStd::string& tag, unsigned int min, unsigned int max)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>AZStd::string</td>
<td>The tag.</td>
</tr>
<tr>
<td>min</td>
<td>unsigned int</td>
<td>The minimum value that can be returned.</td>
</tr>
<tr>
<td>max</td>
<td>unsigned int</td>
<td>The maximum value that can be returned.</td>
</tr>
</tbody>
</table>

**Returns:** A random unsigned integer value within the specified range.

**Return Type:** unsigned int

**Default Return:** 0

**CameraRequestBus**

Provides access to camera properties and services.

**GetFov**

Gets the camera's field of view in degrees

**Syntax**

```cpp
float GetFOV()
```

**Returns:** The camera's field of view as a float.

**Return Type:** float

**Default Return:** s_defaultFoV

**GetNearClipDistance**

Gets the camera's distance from the near clip plane in meters.

**Syntax**

```cpp
float GetNearClipDistance()
```

**Returns:** The camera's distance from the near clip plane as a float in meters.

**Return Type:** float

**Default Return:** s_defaultNearPlaneDistance

**GetFarClipDistance**

Gets the camera's distance from the far clip plane in meters.
Syntax

```lua
float GetFarClipDistance()
```

**Returns:** The camera's distance from the far clip plane as a float in meters.

**Return Type:** float

**Default Return:** s_defaultFarClipPlaneDistance

---

**GetFrustumWidth**

Gets the camera frustum's width.

Syntax

```lua
float GetFrustumWidth()
```

**Returns:** The camera frustum's width as a float.

**Return Type:** float

**Default Return:** s_defaultFrustumDimension

---

**GetFrustumHeight**

Gets the camera frustum's height.

Syntax

```lua
float GetFrustumHeight()
```

**Returns:** The camera frustum's height as a float.

**Return Type:** float

**Default Return:** s_defaultFrustumDimension

---

**SetFov**

Sets the camera's field of view in degrees.

Syntax

```lua
void SetFov(float fov)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fov</td>
<td>float</td>
<td>The field of view in degrees. Possible values are 0 &lt; fov &lt; 180.</td>
</tr>
</tbody>
</table>

---

**SetNearClipDistance**

Sets the near clip plane to the specified distance from the camera in meters.

Syntax
void SetNearClipDistance(float nearClipDistance)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nearClipDistance</td>
<td>float</td>
<td>The distance from the camera in meters. The value should be small, but greater than 0.</td>
</tr>
</tbody>
</table>

**SetFarClipDistance**

Sets the far clip plane to the specified distance from the camera in meters.

**Syntax**

void SetFarClipDistance(float farClipDistance)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>farClipDistance</td>
<td>float</td>
<td>The distance from the camera in meters.</td>
</tr>
</tbody>
</table>

**SetFrustumWidth**

Sets the camera frustum's width.

**Syntax**

void SetFrustumWidth(float width)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>float</td>
<td>The camera frustum's width.</td>
</tr>
</tbody>
</table>

**SetFrustumHeight**

Sets the camera frustum's height.

**Syntax**

void SetFrustumHeight(float height)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>float</td>
<td>The camera frustum's height.</td>
</tr>
</tbody>
</table>

**MakeActiveView**

Makes the camera the active view.

**Syntax**
void MakeActiveView()

**HttpClientComponentNotificationBus**

Event handler for HTTP requests.

**OnHttpRequestSuccess**

Notifies when an HTTP request is successful.

**Syntax**

```lua
void OnHttpRequestSuccess(int responseCode, AZStd::string responseBody)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>responseCode</td>
<td>int</td>
<td>The response code.</td>
</tr>
<tr>
<td>responseBody</td>
<td>AZStd::string</td>
<td>The body of the response.</td>
</tr>
</tbody>
</table>

**OnHttpRequestFailure**

Sent when an HTTP request failed.

**Syntax**

```lua
void OnHttpRequestFailure(int responseCode)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>responseCode</td>
<td>int</td>
<td>The response code.</td>
</tr>
</tbody>
</table>

**HttpClientComponentRequestBus**

Provides services to make HTTP requests.

**MakeHttpRequest**

Makes an HTTP request.

**Syntax**

```lua
void MakeHttpRequest(AZStd::string url, AZStd::string method, AZStd::string jsonBody)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>AZStd::string</td>
<td>The request URL.</td>
</tr>
<tr>
<td>method</td>
<td>AZStd::string</td>
<td>The HTTP request method.</td>
</tr>
<tr>
<td>jsonBody</td>
<td>AZStd::string</td>
<td>The JSON body of the request.</td>
</tr>
</tbody>
</table>
HMDDeviceRequestBus

HMD device bus used to communicate with the rest of the engine. Every device supported by the engine lives in its own Gem and supports this bus. A device wraps the underlying SDK into a single object for easy use by the rest of the system. Every device created should register with the EBus in order to be picked up as a usable device during initialization by the EBus function BusConnect().

GetTrackingState

Gets the most recent HMD tracking state.

**Syntax**

```cpp
TrackingState* GetTrackingState()
```

**Returns:** The tracking state.

**Return Type:** TrackingState*

**Default Return:** nullptr

RecenterPose

Center the current pose for the HMD based on the current direction in which the viewer is looking.

**Syntax**

```cpp
void RecenterPose()
```

SetTrackingLevel

Set the current tracking level of the HMD. Supported tracking levels are defined in struct TrackingLevel.

**Syntax**

```cpp
void SetTrackingLevel(const AZ::VR::HMDTrackingLevel level)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>AZ::VR::HMDTrackingLevel</td>
<td>The tracking level to use with the current HMD. Possible values: kHead - The sensor reads as if the player is standing. kFloor - The sensor reads as if the player is seated or on the floor.</td>
</tr>
</tbody>
</table>

OutputHMDInfo

Outputs the information about the currently connected HMD (contained in the struct HMDDeviceInfo object) to the console and log file.

**Syntax**

```cpp
void OutputHMDInfo()
```
GetDeviceInfo

Get the device info object for this particular HMD.

**Syntax**

```cpp
HMDDeviceInfo* GetDeviceInfo()
```

**Returns:** A pointer to the current HMD's struct HMDDeviceInfo (p. 3261).

**Return Type:** HMDDeviceInfo*

**Default Return:** nullptr

IsInitialized

Gets whether or not the HMD has been initialized. The HMD has been initialized when it has fully established an interface with its required SDK and is ready to be used.

**Syntax**

```cpp
bool IsInitialized()
```

**Returns:** true if the device has been initialized and is usable; otherwise, returns false.

**Return Type:** bool

**Default Return:** false

ControllerRequestBus

Provides information about HMD device controllers.

GetTrackingState

Returns a TrackingState object that contains tracking info about a connected controller. For more information, see struct TrackingState (p. 3261).

**Syntax**

```cpp
TrackingState* GetTrackingState(ControllerIndex controllerIndex)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>controllerIndex</td>
<td>int</td>
<td>Specify 0 for the left controller; 1 for the right controller.</td>
</tr>
</tbody>
</table>

**Returns:** A pointer to the TrackingState object for the connected controller.

**Return Type:** TrackingState*

**Default Return:** nullptr

IsConnected

Returns whether the specified controller is connected.
Syntax

```cpp
bool IsConnected(ControllerIndex controllerIndex)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>controllerIndex</td>
<td>int</td>
<td>Specify 0 for the left controller; 1 for the right controller.</td>
</tr>
</tbody>
</table>

**Returns:** A Boolean that indicates whether the specified controller is connected.

**Return Type:** bool

**Default Return:** false

**VideoPlaybackRequestBus**

Provides access to video playback services.

**Play**

Start or resume playing a movie that is attached to the current entity.

**Syntax**

```cpp
void Play()
```

**Pause**

Pause a movie that is attached to the current entity.

**Syntax**

```cpp
void Pause()
```

**Stop**

Stop playing a movie that is attached to the current entity.

**Syntax**

```cpp
void Stop()
```

**EnableLooping**

Set whether or not the movie attached to the current entity loops.

**Syntax**

```cpp
void EnableLooping(bool enable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>bool</td>
<td>Specify true to loop; false to not loop.</td>
</tr>
</tbody>
</table>
**IsPlaying**

Returns whether or not the video is currently playing

**Syntax**

```lua
bool IsPlaying()
```

**Returns:** true if the video is currently playing; false if the video is paused or stopped.

**Return Type:** bool

**Default Return:** false

**SetPlaybackSpeed**

Sets the playback speed based on a factor of the current playback speed.

**Syntax**

```lua
void SetPlaybackSpeed(float speedFactor)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>speedFactor</td>
<td>float</td>
<td>The speed modification factor to apply to playback speed. For example, specify 0.5f to play at half speed or 2.0f to play at double speed.</td>
</tr>
</tbody>
</table>

**VideoPlaybackNotificationBus**

This bus contains event handlers for video playback services.

**OnPlaybackStarted**

Event that fires when the movie starts playback.

**Syntax**

```lua
void OnPlaybackStarted()
```

**OnPlaybackPaused**

Event that fires when the movie pauses playback.

**Syntax**

```lua
void OnPlaybackPaused()
```

**OnPlaybackStopped**

Event that fires when the movie stops playback.

**Syntax**

```lua
void OnPlaybackStopped()
```
OnPlaybackFinished

Event that fires when the movie completes playback.

Syntax

```c
void OnPlaybackFinished()
```

Using Script Events

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Scripting in Lumberyard is designed around event driven paradigms. Rather than directly accessing information from a given entity or one of its components, you use events to send and receive information and take action in a decoupled environment.

Like entities and components, scripts can use events to communicate with each other. These events are called script events.

To author script events, use the Asset Editor or Lua. Script Canvas and Lua can send or receive the events that you create. In Script Canvas, you can add nodes that send or receive script events. Events sent from Script Canvas can be handled in Lua. Likewise, events sent from Lua can be handled in Script Canvas.

Prerequisites

To use script events, you must enable the Script Events gem in your project. For information about enabling gems, see Enabling Gems (p. 1064).

Topics

- Creating Script Events in Script Canvas (p. 2762)
- Using Script Events in Script Canvas (p. 2765)
- Using Script Events in Lua (p. 2766)
- Script Events Best Practices (p. 2769)

Creating Script Events in Script Canvas

In Lumberyard Editor, you can create script events by using the Asset Editor.

To create a script event

1. In Lumberyard Editor, choose Tools, Asset Editor.
2. In the Asset Editor, choose File, New, Script Events.
3. In the **Asset Editor**, enter the information that defines the script event.

<table>
<thead>
<tr>
<th>Script Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a name that identifies the script event in Lua and in the Script Canvas editor <strong>Node Palette</strong>. Event names must have only alphanumeric characters and cannot start with a number or contain white space.</td>
</tr>
</tbody>
</table>
Creating Script Events in Script Canvas

<table>
<thead>
<tr>
<th>Script Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category</strong></td>
<td>Enter a name for the category in the Script Canvas editor <strong>Node Palette</strong> in which the script events appear. The default category is <strong>Script Events</strong>. To nest categories, use the syntax <code>category/sub_category/sub_category</code>.</td>
</tr>
<tr>
<td><strong>Tooltip</strong></td>
<td>Enter a description for the script event. The description appears when the user pauses a pointer on the script event in the <strong>Node Palette</strong> or on the event node in a graph.</td>
</tr>
<tr>
<td><strong>Address Type</strong></td>
<td>(Optional). Choose a data type for the value that addresses this script event. Possible types are <strong>String</strong>, <strong>Entity Id</strong>, or <strong>Tag</strong>.</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>Specify a list of events that are sent and received by the scripting system. Define the events in the same way that you define functions in a programming language. The events that you create are available for dragging and dropping from the Script Canvas <strong>Node Palette</strong>.</td>
</tr>
</tbody>
</table>

**Event Properties**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>The name of the event.</td>
</tr>
<tr>
<td><strong>Tooltip</strong></td>
<td>Enter the description for the event that appears when the user pauses a pointer on the event name.</td>
</tr>
<tr>
<td><strong>Return value type</strong></td>
<td>(Optional). Choose the data type of the value that the handler of the event returns to the sender of the event. If Return Type is not None, a value of the specified type must be connected from the receiver node to the sender node.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>Click the plus (+) icon to add one or more parameters for the event function.</td>
</tr>
</tbody>
</table>

**Parameter Properties**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Enter the name of a function parameter.</td>
</tr>
<tr>
<td><strong>Tooltip</strong></td>
<td>Enter the description for the parameter that appears when the user pauses a pointer on the parameter name.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Choose the data type of the parameter.</td>
</tr>
</tbody>
</table>

4. Choose **File**, **Save**, or press **Ctrl+S**.
5. In the **Save As** dialog box, enter a file name for your `.scriptevents` file, and click **Save**.

**Note**

To ensure that the script event asset saved correctly, check the bottom of the Asset Editor for the message "file_name.scriptevents - Asset loaded!" message. If you see the error message Failed to save asset due to validation error, check the Lumberyard Editor **Console** window or the `lumberyard_version\dev\Cache\project_name\pc\user\log\Editor.log` file for more information.
Using Script Events in Script Canvas

Script Canvas scans for and detects script event assets, so after you define a script event, you can use it in Script Canvas.

Script Events in the Node Palette

Script event assets appear by default in the Script Events category in the Script Canvas editor Node Palette.

Note
To change the name of the category, open the script event's asset definition in the Asset Editor and edit the Category property.

Sending Events

You can send an event by adding a Send method_name node to a Script Canvas graph.

To send an event

1. Drag and drop the method that you want to send onto the Script Canvas graph.
2. In the context menu, choose Send method_name.

A method send node is added to the graph.

3. Connect this node to the appropriate logic and data inputs. When the Script Canvas graph runs, it sends the event to the entities or systems to which the node is connected.
Handling Events

You can handle an event by adding a Receive `method_name` node to a Script Canvas graph.

To handle an event

1. Drag and drop a script event method onto the canvas.
2. In the context menu, choose Receive `method_name`.

An event handler method node is added to the graph.

3. Connect your event handling logic to the Out pin of the node.
4. Connect the data pin as needed.

Using Script Events in Lua

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lua scripts can use script events to communicate with each other. There are two example scripts that show this communication, both available in the `lumberyard_version\dev\Gems\ScriptEvents\Assets\Scripts\Example` directory. They are called `ScriptEvents_Addressable.lua` and `ScriptEvents_Broadcast.lua`. If an EBus is addressed, events are sent to a specific address ID. Events that are broadcast globally are received at all addresses. For more information, see Working with the Event Bus (EBus) system (p. 1851).

`ScriptEvents_Addressable.lua`
The `ScriptEvents_Addressable.lua` example script implements a handler for a script event that requires an address for a handler to be invoked. It broadcasts a method, but only handlers connected to the address that match the one specified in the event can invoke it.

**Example**

```lua
-- ScriptEvents_Addressable.lua
function ScriptTrace(txt)
    Debug.Log(txt)
end

function ScriptExpectTrue(condition, msg)
    if (not condition) then
        ScriptTrace(msg)
    end
end

-- This example shows how to implement a handler for a Script Event that requires an address
-- in order for a handler to be invoked.
luaScriptEventWithId = {
    MethodWithId0 = function(self, param1, param2)
        ScriptTrace("Handler: " .. tostring(param1) .. " " .. tostring(param2))
        ScriptExpectTrue(typeid(param1) == typeid(0), "Type of param1 must be " .. tostring(typeid(0)))
        ScriptExpectTrue(typeid(param2) == typeid(EntityId()), "Type of param2 must be " .. tostring(typeid(EntityId())))
        ScriptExpectTrue(param1 == 1, "The first parameter must be 1")
        ScriptExpectTrue(param2 == EntityId(12345), "The received entity Id must match the one sent")
        ScriptTrace("MethodWithId0 handled")
        return true
    end,
    MethodWithId1 = function(self)
        ScriptTrace("MethodWithId1 handled")
    end
}

-- "Script_Event" will be the name of the callable Script Event. It requires the address type to be a string.
luaScriptEventDefinition = ScriptEvent("Script_Event", typeid("") -- Event address is of string type

-- Define some methods for handlers to implement.
luaScriptEventDefinition:AddMethod("MethodWithId0", typeid(false)) -- Return value is Boolean
method0:AddParameter("Param0", typeid(0))
method0:AddParameter("Param1", typeid(EntityId()))

-- NOTE: Types are specified using the typeid keyword with a VALUE of the type you want. For example, typeid("EntityId")
-- produces the type id for a string, not the EntityId type.
luaScriptEventDefinition:AddMethod("MethodWithId1") -- No return, no parameters.
```
-- After the Script Event is defined, call Register to enable it. Typically this should be done within the OnActivate method.
scriptEventDefinition:Register()

-- At this point, the script event is usable. Use the Script_Event.Connect method to connect a handler to it. The following example
-- uses luaScriptEventWithId as the handler that implements the methods defined earlier. Notice that the call connects using
-- the string "ScriptEventAddress" as the address for this event. Methods sent to a different address will not be handled by
-- this handler.
scriptEventHandler = Script_Event.Connect(luaScriptEventWithId, "ScriptEventAddress")

-- Now invoke the event and specify "ScriptEventAddress" as the address. The handler previously
-- connected will be able to handle this event.
local returnValue = Script_Event.Event.MethodWithId0("ScriptEventAddress", 1,
EntityId(12345))

-- "Method0" should return true. Verify the result.
ScriptExpectTrue(returnValue, "Method0's return value must be true")

-- Finally, send MethodWithId1. This method does not require any parameters, but does require the address.
ScriptEvento.MethodWithId1("ScriptEventAddress")

ScriptEvents_Broadcast.lua

The ScriptEvents_Broadcast.lua example script implements a handler for a broadcast script event. Because broadcast script events do not specify an address type, any handler can connect to them.

Example

-- ScriptEvents_Broadcast.lua
function ScriptTrace(txt)
    Debug.Log(txt)
end

function ScriptExpectTrue(condition, msg)
    if (not condition) then
        ScriptTrace(msg)
    end
end

-- This example shows how to implement a handler for a broadcast script event.
-- Because broadcast script events do not specify an address type, they can be handled
-- by simply connecting to them.
luaScriptEventBroadcast = {

    -- The following method is called as a result of a broadcast call on the script event.
    BroadcastMethod0 = function(self, param1, param2)
        ScriptTrace("Handler: ", tostring(param1) .. " ", tostring(param2))
        ScriptExpectTrue(typeid(param1) == typeid(0), "Type of param1 must be ":tostring(typeid(0)))
        ScriptExpectTrue(typeid(param2) == typeid(EntityId()), "Type of param2 must be ":tostring(typeid(EntityId())))
        ScriptExpectTrue(param1 == 2, "The first parameter must be 2")
        ScriptExpectTrue(param2 == EntityId(23456), "The received entity Id must match the one sent")
        ScriptTrace("BroadcastMethod0 Called")
    end

    BroadcastMethod1 = function(self, param1, param2)
        ScriptTrace("Handler: ", tostring(param1) .. " ", tostring(param2))
        ScriptExpectTrue(typeid(param1) == typeid(0), "Type of param1 must be ":tostring(typeid(0)))
        ScriptExpectTrue(typeid(param2) == typeid(EntityId()), "Type of param2 must be ":tostring(typeid(EntityId())))
        ScriptExpectTrue(param1 == 2, "The first parameter must be 2")
        ScriptExpectTrue(param2 == EntityId(23456), "The received entity Id must match the one sent")
        ScriptTrace("BroadcastMethod1 Called")
    end

    BroadcastMethod2 = function(self, param1, param2)
        ScriptTrace("Handler: ", tostring(param1) .. " ", tostring(param2))
        ScriptExpectTrue(typeid(param1) == typeid(0), "Type of param1 must be ":tostring(typeid(0)))
        ScriptExpectTrue(typeid(param2) == typeid(EntityId()), "Type of param2 must be ":tostring(typeid(EntityId())))
        ScriptExpectTrue(param1 == 2, "The first parameter must be 2")
        ScriptExpectTrue(param2 == EntityId(23456), "The received entity Id must match the one sent")
        ScriptTrace("BroadcastMethod2 Called")
    end

}
return true
end,

BroadcastMethod1 = function(self)
    ScriptTrace("BroadcastMethod1 Called")
end
}

local scriptEventDefinition = ScriptEvent("Script_Broadcast") -- Script_Broadcast is the
-- name of the callable script event.
-- Define methods for Script_Broadcast
local method0 = scriptEventDefinition:AddMethod("BroadcastMethod0", typeid(false)) -- To
-- add a method, provide a method name and an optional return type.
method0:AddParameter("Param0", typeid(0))
method0:AddParameter("Param1", typeid(EntityId()))

-- NOTE: Types are specified using the typeid keyword with a VALUE of the type you want. For example, typeid("EntityId")
-- produces the type id for a string, not the EntityId type.

scriptEventDefinition:AddMethod("BroadcastMethod1")
-- After the Script Event is defined, call Register to enable it. Typically this should be
done within the OnActivate method.
scriptEventDefinition:Register()

-- At this point, the script event is usable. Use the Script_Broadcast.Connect method to
-- connect a handler to it. The following example
-- uses luaScriptEventWithId as the handler that implements the methods defined earlier.
scriptEventHandler = Script_Broadcast.Connect(luaScriptEventBroadcast)

-- To test the event, broadcast BroadcastMethod0. As defined, the method expects two
parameters and returns a boolean value.
local returnValue = Script_Broadcast.Broadcast.BroadcastMethod0(2, EntityId(23456))

-- BroadcastMethod0 should return true. Verify the result.
ScriptExpectTrue(returnValue, "BroadcastMethod0's return value must be true")

-- To broadcast an event without a return or parameters, invoke BroadcastMethod1.
Script_Broadcast.Broadcast.BroadcastMethod1()

---

Script Events Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following are some best practices for using script events.

Ensure Entities are Activated Before Events are Sent

Sending events during entity activation can have undesired results. Because the order of activation of entities is not guaranteed, when an event is sent during activation, some entities that need to handle the event might not receive it. In particular, the On Graph Start and On Entity Activated events are subject to activation order issues.

To ensure that all entities that need to listen for and handle a given script event are ready to receive the event, queue the message on the tick bus (p. 1019). One way to implement this strategy in Script
Canvas is to connect the **On Graph Start** node to a **Tick Delay** node. The delay helps to ensure that when a script event message is sent, all entities that could possibly be connected to that script event receive the event.

---

**Be Aware of Script Event Asset Versioning**

Because Script Events are user-created assets, problems can occur when an asset that is referenced in existing scripts or Script Canvas graphs changes.

**Note**

Script Canvas provides Script Event version validation. When a Script Event asset is modified, Script Canvas updates the Script Event nodes that reference it. If you open a graph that has a Script Event that has been modified, the graph is marked as modified. To update the Script Event nodes to their latest versions, save the graph.

**Use Script Events Instead of the Gameplay Notification Bus System**

Script events offer the following advantages over the gameplay bus (p. 1030) system. Script events:

- Are data driven.
- Support more data types.
- Require less maintenance.
- Can be used in both Script Canvas and Lua.

For these reasons, script events supersede the gameplay notification bus system and the following GameplayNotificationBus related classes:

- GameplayNotificationBus
- GameplayNotificationId
- BehaviorGameplayNotificationBusHandler
- GameplayEventHandlerNode (Legacy)

**Migrating to Script Events**

If your project uses GameplayNotificationBus, you can modify it to use script events.
To migrate from GameplayNotificationBus to script events

1. Create a new Script Event that performs the event messaging that you require.
2. Identify the Script Canvas graphs or Lua scripts that use GameplayNotificationBus.
3. Replace the nodes or code that use GameplayNotificationBus with either a Script Event Send or a Script Event Receive.
Simulating physics behavior with the PhysX system

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard’s PhysX system acts upon entities to create realistic physical effects such as collision detection and rigid body dynamics simulation. To use the PhysX system, install the PhysX SDK using the Lumberyard Setup Assistant (p. 16).

**Note**
Lumberyard’s PhysX system does not interact with Physics (Legacy) (p. 534) component entities or the legacy Physics.

**Topics**
- PhysX Gems (p. 2772)
- PhysX Components (p. 2773)
- PhysX Configuration (p. 2773)
- PhysX Materials (p. 2773)
- PhysX Debugging (p. 2774)
- Configuring the PhysX System (p. 2774)
- Physics materials (p. 2797)
- PhysX Scene Queries (p. 2804)
- Dynamic joints with PhysX (p. 2813)
- Create global or localized wind forces with PhysX (p. 2822)
- Debugging PhysX (p. 2826)
- PhysX Best Practices (p. 2832)
- Simulated destruction with NVIDIA Blast (p. 2833)
- Simulate cloth with NVIDIA Cloth (p. 2857)

**PhysX Gems**

The PhysX system uses the following gems, which you can enable (p. 1064) in the Project Configurator:

- **PhysX (p. 1188)** – Provides integration for the NVIDIA PhysX SDK into Lumberyard. The integration provided includes a suite of components, configuration via the editor, Script Canvas integration, PhysX Visual Debugger integration, and a simplified API abstraction layer for games.

  For more information, see PhysX (p. 1188).

- **PhysX Characters (p. 1188)** – Provides integration for character controllers and ragdolls. To enable the PhysX Characters gem, you must first enable the PhysX (p. 1188) gem.

  For more information, see PhysX Characters (p. 1188).
• **PhysX Debug (p. 1188)** – Provides debug visualizations of PhysX scene geometry that you can enable with console commands and other tools.

For more information, see PhysX Debug (p. 1188).

### PhysX Components

The **PhysX** gem has the following components, which you can add (p. 479) to entities by using the Entity Inspector (p. 475):

- **PhysX Collider (p. 735)** – Enables physics objects to collide with other physics objects. An entity that does not have a PhysX Rigid Body Physics component is a static collider, while an entity with the component is a dynamic collider.

- **PhysX Force Region (p. 746)** – Enables an entity to specify a region that applies physical force to entities. For each physics simulation frame, the component applies force to entities that are in the bounds of the region.

- **PhysX Rigid Body (p. 759)** – Enables an entity to be simulated by physics. Rigid body mode can be kinematic or dynamic. Dynamic rigid bodies respond to collision events with other rigid bodies. Kinematic rigid bodies are not affected by outside forces and gravity; their motion is driven by scripting.

- **PhysX Terrain (p. 767)** – Implements physical interaction with the terrain. It exports terrain and saves it as an asset that loads at runtime.

The **PhysX Characters** gem has the following components:

- **PhysX Character Controller (p. 729)** – Implements basic character interactions with the physical world. For example, it can control interactions with slopes and steps, manage interactions with other characters, and prevent characters from walking through walls or passing through terrain.

- **PhysX Ragdoll (p. 757)** – Enables animation of certain character behaviors. The physical representation is usually a hierarchical collection of rigid bodies with simple shapes connected by joints.

You can use the following video tutorial to learn about PhysX components in Lumberyard.

Nvidia PhysX Lumberyard Integration: Adding the Terrain, Collider, and Rigid Body Components

### PhysX Configuration

Use the **PhysX Configuration** window in Lumberyard Editor to configure global settings, collision layers, collision groups, and PhysX Visual Debugger settings.

For more information, see Configuring the PhysX System (p. 2774).

You can use the following video tutorial to learn about the collision layers and groups in Lumberyard.

NVIDIA PhysX Lumberyard Integration: Using Collision Filtering

### PhysX Materials

PhysX materials allow simulation properties to be configured by entity. Materials customize how an object reacts when it hits a surface and control qualities like friction and bounciness. You use the Asset
Editor to create a material library, assign the library to a collider, and then select a specific material from the library for the collider.

For more information, see Physics materials (p. 2797).

You can use the following video tutorial to learn about the PhysX materials in Lumberyard,

NVIDIA PhysX Lumberyard Integration: Physical Material Creation

# PhysX Debugging

To verify the implementation of interactions in the simulated world, the following tools are available.

- **PhysX Debug gem** – The PhysX Debug gem is recommended if you are a developer or technical artist. You can use this tool to view the physics world in real time in Lumberyard Editor's editor mode or game mode. To activate the tool, you use console commands or an immediate mode graphical user interface (ImGui). The tool displays PhysX debug lines within the editor and game modes.

  For more information, see PhysX Debug (p. 1188).

- **PhysX Visual Debugger** – The PhysX Visual Debugger (PVD) is a third party tool provided by NVIDIA that is useful for deep inspection of the PhysX world. Lumberyard can connect PhysX worlds and scenes to a running PVD application instance. You can use the PVD to step through your simulation and examine various properties at your own pace in detail.

  For information on configuring Lumberyard's connection to PVD, see Debugger Configuration (p. 2790).

For more information, see Debugging PhysX (p. 2826).

# Configuring the PhysX System

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The PhysX (p. 1188) gem can be configured for each project for your game. Your settings are saved in your project's `lumberyard_version\dev\project_name\default.physxconfiguration` file. If you use a version control system, include this file.

To configure PhysX settings, use the PhysX Configuration tool.

**To open the PhysX Configuration tool**

- In Lumberyard Editor, choose Tools, PhysX Configuration.

Topics
- Global Configuration (p. 2775)
- Collision Layers (p. 2785)
- Collision Groups (p. 2787)
- Debugger Configuration (p. 2790)
- PhysX World Programming Notes (p. 2796)
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

On the **Global Configuration** tab, you can configure world settings and editor settings.

![Global Configuration tab](image)

**World Configuration**

The following table describes the settings for **World Configuration**.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Time Step</td>
<td>Specifies the largest time step that the simulation can process. This setting prevents instability in the simulation if m_fixedTimeStep is not large enough.</td>
</tr>
<tr>
<td>Fixed Time Step</td>
<td>Specifies the fixed time step that the simulation can process. This setting prevents instability in the simulation if m_fixedTimeStep is not large enough.</td>
</tr>
</tbody>
</table>
**Description**

set.
It also prevents the simulation from taking too long.

If the time between frames is greater than \( m_{\text{maxTimeStep}} \), the time is limited to this value. Specify small increments.

The default is \( 0.05 \) (1/20th of a second).
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Time Step</td>
<td>Controls the frequency of the fixedTimeStep interval at which Lumberyard does the simulation. The default is 0.017 (1/60th of a second).</td>
</tr>
</tbody>
</table>

**Note**

- A lower value results in a more accurate simulation, but at a higher runtime cost.
- Higher values can cause results that are less stable.
- If this value is set to 0, the
simulation does not use the fixed time and uses the time between frames, which can vary.

- If the frame time is greater than this value, Lumberyard splits the time to the number of steps yielded by the following calculation:

\[
\frac{\text{frame time}}{\text{m_fixedTimeStep}}
\]
**Global Configuration**

**Property**  
**Description**

**Gravity**
- Gravity vector in the world.
- The default X, Y, and Z values are 0.0, 0.0, and -9.81.

**Raycast Buffer Size**
- Maximum number of hits that can be returned from a raycast query.
- The default is 32.

**Shapecast Buffer Size**
- Maximum number of hits that can be returned from a shapecast query.
- The default is 32.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlap Query Buffer Size</td>
<td>Maximum number of hits that can be returned from an overlap query. The default is 32.</td>
</tr>
<tr>
<td>Continuous Collision Detection</td>
<td>Enables continuous collision detection in the world. Disabled by default.</td>
</tr>
</tbody>
</table>
**Persistent Contact Manifold**

to improve the calculation of the entities that are colliding and preserves this data between frames.

Enabled by default, which is the recommended setting.

Persistent manifolds store the contact data created in one frame for potential reuse in subsequent frames. For example, if the corner of box A collides with
the side of box B, the contact data is stored and the manifold stores the collision point. If box A's corner no longer collides in approximately in the same location in the next frame, the data is disregarded.

If box A moves too far (for example, if it rotates so far that the corner
no longer touches), that data is discarded. Otherwise, the manifold provides faster collision results by using the data from the previous frame to calculate the collision. Although persistent manifolds require more memory, they improve performance and the accuracy of the simulation by reducing jitter and other unwanted physics artifacts.
Editor Configuration

The following options relate to the Debug Draw COM (center of mass) option of the PhysX Rigid Body (p. 759) component.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug Draw</td>
<td>Debug Draw Center of Mass Color</td>
</tr>
<tr>
<td>Center Size</td>
<td>The size of the debug draw circle that represents the center of the mass.</td>
</tr>
<tr>
<td></td>
<td>Possible values are from 0.1 to 5.0 meters.</td>
</tr>
<tr>
<td></td>
<td>The default value is 0.1.</td>
</tr>
<tr>
<td>Color</td>
<td>The color of the debug draw circle that represents the center of the mass.</td>
</tr>
<tr>
<td></td>
<td>To specify a color, enter its RGB values.</td>
</tr>
</tbody>
</table>
Collision Layers

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use collision layers to group objects of the same type. Use collision groups to define what the collision layers collide with. You can have a maximum of 64 layers per game.

Common types of layers are objects like the following:

- Terrain
- Static objects
- Players
- Enemies
- Projectiles

The layers that you define are specific to your game. When a collider is created, it is assigned a collision layer with index [0] called Default. This can't be changed.
To create a collision layer

1. In Lumberyard Editor, choose **Tools, PhysX Configuration**.
2. Click the **Layers** tab.
3. Type the name of the layer into an available text field. Layer names must be 32 characters or less.

![PhysX Configuration (PREVIEW)](image)

To assign a collision layer to a collider

1. In the Lumberyard Editor viewport, create an entity and select it.
2. In the **Entity Inspector**, add a **PhysX Collider** component to the entity.
3. In the **PhysX Collider** component, for **Collision Layer**, choose one of the colliders that you created from the drop-down list.

![PhysX Collider](image)
Notes

- If you rename a layer, its references are updated automatically, but you can't reorder layers.
- If an entity with a PhysX Collider component is selected in the viewport and you use the PhysX Configuration window to create, rename, or delete a collision layer, the changes do not appear in the Entity Inspector. To see the changes, deselect and reselect the entity in the viewport.

Collision Groups

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use collision groups to define the layers that a collider interacts with. A collision group is similar to a mask with specific bits set in which each bit corresponds to a collision layer.

Two colliders interact if their collision layers are in each other's collision group. If one collision layer is not present in the other layer's collision group, the colliders don't interact.

To create a collision group

1. In Lumberyard Editor, choose Tools, PhysX Configuration.
2. Click the Collision Groups tab. The layers that are configured on the Layers tab appear here as columns.
3. Click Add, and then enter the name of the group into the text box.
4. Select or clear the check boxes to specify the layers to include in each collision group.
5. Close the configuration tool. The configuration tool must be closed for changes to the groups to take effect.

To assign a collision group to a collider

1. In the Lumberyard Editor viewport, select the entity that has the collider.
2. In the Entity Inspector, in the PhysX Collider component, for Collides With, choose one of the collision groups that you created from the drop-down list.
Example Collision Group Configuration

The following example defines Player, Enemy, Bullet, Foliage, and Terrain layers, and the following collision groups:

- **PlayerBullet** – Collides with Enemy and Terrain.
- **EnemyBullet** – Collides with Player and Terrain.
- **TerrainCollision** – Collides with Player, Enemy, Bullet, and Terrain.
- **PlayerCollision** – Collides with Enemy, Bullet, and Terrain.

A bullet fired by the player has the following layer and group:

- **Layer**: Bullet
- **Group**: PlayerBullet
A bullet fired by the enemy has the following layer and group:

- **Layer:** Bullet
- **Group:** EnemyBullet

**Note**
You don't have to define an "enemy bullet" or a "player bullet" layer. Instead, you have a single Bullet layer and create separate collision groups to specify the objects that it collides with.

### Creating Collision Layers and Groups Programmmatically

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The API for collision layers and groups is defined in the Collision.h and CollisionBus.h headers. The following code shows example include statements.

```cpp
#include <AzFramework/Physics/Collision.h>
#include <AzFramework/Physics/CollisionBus.h>
// For the rest of the examples:
using namespace AzFramework;
using namespace Physics;
```

For source code, see the `lumberyard_version\dev\Code\Framework\AzFramework\AzFramework\Physics\Collision.*` files.

### Retrieving Layers and Groups

You can use the `layer(layer_name)` and `group(group_name)` methods to retrieve instances of collision layers and groups that you created in Lumberyard Editor (p. 2787). These methods perform a lookup of the definitions defined in the PhysX Configuration tool. If no collision layer is found that matches the name, the default layer (0) is returned.

```cpp
CollisionLayer layer("MyLayer");
CollisionGroup group("MyGroup");
```

You can also use a bus to look up layers and groups, as in the following code:

```cpp
CollisionLayer layer;
CollisionRequestBus::BroadcastResult(layer, &Physics::CollisionRequests::GetCollisionLayerByName, layerName);
CollisionGroup group;
CollisionRequestBus::BroadcastResult(group, &Physics::CollisionRequests::GetCollisionGroupByName, groupName);
```

### Creating Collision Groups in Code

Unlike the predefined collision groups that you create in Lumberyard Editor, you can use code to create collision groups at runtime. This is useful when you don’t know the layers involved in advance at therefore must define a collision group at runtime.
The following example code creates a collision group at runtime that contains an Enemy layer and a Tree layer. This handles the situation in which the identity of the tree and the enemy are not known prior to the start of gameplay.

```
CollisionLayer layer1("Enemy"), layer2("Tree");
CollisionGroup group = AzFramework::Physics::CollisionGroup::None;
group.SetLayer(layer1, true);
group.SetLayer(layer2, true);
```

If all the layers required to construct the collision group are known at the same time, you can use overloaded operators, as in the following example:

```
CollisionGroup group = CollisionLayer("Layer1") | CollisionLayer("Layer2") |
CollisionLayer("Layer3");
```

## Debugger Configuration

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.

Download O3DE or visit the AWS Game Tech blog to learn more.

In the PhysX Configuration tool, you can specify how to interact with the PhysX Visual Debugger (PVD). The PVD is a third-party application that records your PhysX data from Lumberyard Editor. You can then review this data to see how your physics effects appear.

For more information, see the NVIDIA PhysX Visual Debugger (PVD) documentation.

**To configure the PhysX Visual Debugger**

1. To get started, download the PVD.

   **Note**
   You must have a NVIDIA account to download the PVD. If you don’t already have an account, follow the steps to create and then sign in your account.

2. Follow the installation steps.
3. After you install the application, open the PVD. This application must be running if you want to record data from Lumberyard Editor.
4. In Lumberyard Editor, open a level or create one that has entities with PhysX components. For example, you can create a dynamic entity that falls. For more information, see Creating a Dynamic PhysX Entity (p. 762).
5. In Lumberyard Editor, choose Tools, PhysX Configuration.
6. Click the Debugger tab.
7. You can specify the following settings.
<table>
<thead>
<tr>
<th>Description</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PVD Host</strong></td>
<td>127.0.0.1</td>
</tr>
<tr>
<td><strong>PVD Port</strong></td>
<td>5425</td>
</tr>
<tr>
<td><strong>PVD Timeout</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>PVD Transport Type</strong></td>
<td>Network</td>
</tr>
<tr>
<td><strong>PVD Auto Connect</strong></td>
<td>Disabled</td>
</tr>
<tr>
<td><strong>PVD Reconnect</strong></td>
<td></td>
</tr>
</tbody>
</table>

**PhysXption**

**PVD Settings**

**PVD Host**
address of the PVD. The default is 127.0.0.1 (localhost).

**PVD Port**
the PVD monitors. The default is 5425.
### PhysX Option

#### PVD Settings

<table>
<thead>
<tr>
<th>PVD Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
<td>Timeout, in seconds, used to connect to the PVD. The default is 10.</td>
</tr>
</tbody>
</table>
PhysX

PVD Settings

Transport Type

Supports writing to a TCP/IP network or a file. You can specify the following options:

- **Network**
  - All scene information is sent through the network to the debugger.

- **File**
  - All scene information is written to a file. The file can be loaded into the debugger offline.

The default
**PhysX Option**

**PVD Settings** is **Network**.

**PVD** automatically **Autoconnects** the **Lumberyard Editor** to the PhysX Visual Debugger.

You can specify the following options:

- **Disabled**
- **Editor**
- **Game**

To set this option, you must enable the **PhysX Debug** (p. 1188) gem.
<table>
<thead>
<tr>
<th>PVD Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconnect</td>
<td>Specifies whether to reconnect when switching between editor and gameplay mode.</td>
</tr>
</tbody>
</table>

Enabled by default.

As a best practice, leave this setting enabled. If you clear this option, the PhysX Visual Debugger continues to record data. This can impact performance.

To set this option, you must enable the PhysX Debug (p. 1188) gem.
8. To verify that the PVD is connected to Lumberyard, for **PVD Auto Connect**, choose **Game** or **Editor** and then enter gameplay or editor mode. Depending on what you choose, the following message appears in the console.

   (PhysX) – Successfully connected to the PhysX Visual Debugger (PVD).

9. Open the PhysX Visual Debugger to view the recorded information.

**Example**

![PhysX Visual Debugger](image)

10. You can also manually connect or disconnect from the PVD using the following console variable commands.

   ```
   physx_PvdConnect
   physx_PvdDisconnect
   ```

   For more information, see *Debugging PhysX (p. 2826).*

### PhysX World Programming Notes

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
For physics objects to be simulated, they must exist inside a world. Multiple worlds can have uses like the following:

- To simulate the result of an action in the first world. For example, the second world might show what a tower of blocks might look like five seconds from now if it were knocked over in the first world.
- To simulate a subset of objects that you don't want to interact with the rest of the world. For example, you could simulate the movement of objects attached to a player's belt.
- To overcome hardware or software limits on a single large world. By tiling the single world into multiple smaller worlds and moving objects among them, you can create the illusion of a single large world.

The PhysX gem automatically creates a world inside ActionGame with the default ID of AZPhysicalWorld. By default, all objects are added to this world and simulated each frame. You can find the source code in the following directory: lumberyard_version\dev\Gems\CryLegacy\Code\Source\CryAction\ActionGame.*

World ID

Every world created in the PhysX gem is addressable by an ID of type AZ::Crc32. Use this ID to address the WorldRequestBus.

If you have a single world in your game, you can use BroadcastResult to invoke WorldRequestBus, as in the following example:

```c++
// Single world setup
RayCastHit choose;
WorldRequestBus::BroadcastResult(choose, &WorldRequests::RayCast, request);
```

If you have a multi-world game, use EventResult and pass in the world ID, as in the following example:

```c++
// Multiple world setup
RayCastHit choose;
WorldRequestBus::EventResult(choose, AZ_CRC("AZPhysicalWorld"), &WorldRequests::RayCast, request);
```

**Note**

If your game creates multiple worlds, it must manage the objects that are added into those worlds.

### Step Constants

You can configure step constants for Physics::WorldSettings when PhysXWorld is created. For more information, see World Configuration (p. 2775).

## Physics materials

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Physics materials define how a PhysX collider reacts to collisions through properties like friction and restitution (bounce). In Lumberyard, you specify physics materials for each PhysX collider and store
them inside a physics material library that you create with the Asset Editor. One library can store all the physics materials for a project or separate libraries can be made for different physics material types.

**Tip**
Having one library makes it easier to see all the physics materials and their properties in a project.

**Topics**
- Physics material properties (p. 2798)
- Create a physics material (p. 2799)
- Assign a physics material library (p. 2800)
- Assign a physics material (p. 2802)
- Assign physics materials per face (p. 2802)
- Terrain physics materials (p. 2803)

### Physics material properties

![Asset Editor](image)

**Surface Type**
Name of the physics material.

**Dynamic Friction**
The friction coefficient when the PhysX collider is moving.

\[0.0\]: No friction.

**Static Friction**
The friction coefficient when the PhysX collider is still.
Create a physics material

Physics materials define the physical properties of PhysX colliders. To create physics materials, you first create a physics material library.

To create a physics material

1. Choose Asset Editor from the Tools menu.
2. In the Asset Editor, choose New, Physics Material from the File menu to create a new physics material library.
3. Create a physics material in the library by clicking the + on the right.

4. Name the physics material by editing the **Surface Type** property. Set the material properties as desired.

   **Note**
   You can add additional physics materials to the library by repeating the preceding two steps.

5. Choose **Save As** from the **File** menu in the Asset Editor to save the library and physics material.

### Assign a physics material library

Physics materials can be assigned to entire PhysX colliders, or on a per-face basis if the PhysX mesh is a triangle mesh. To assign a material, you first assign the library containing the desired physics material.
To assign a physics material library

1. Your project has a default physics material library. When new PhysX collider components are added to entities, the default physics material library is assigned to the collider. To change the default physics material library, choose PhysX Configuration from the Tools menu. In the Global Configuration tab, click the ... button to the right of Default Physics Material Library to change the default library.

![PhysX Configuration Screen](image)

Note
The default physics material library cannot be removed. If the default physics material library is deleted or invalid, a new library named SurfaceTypeMaterialLibrary is created automatically when the editor is opened.

2. Newly added PhysX Collider components have the default physics material library assigned. A default physics material library assignment appears in gray text in the component properties. To change the physics material library assigned to a collider, click the ... button to the right of Physics Material - Library in the PhysX Collider component properties. To reset to the default physics material library, click the x button to the right of Physics Material - Library in the PhysX Collider component properties.

![PhysX Collider Screen](image)
Assign a physics material

When the Physics Asset shape is selected, and Physics Materials from Mesh is enabled, the physics material for this collider is automatically set based on the surface type in the mesh's material. To manually set the physics material, uncheck the Physics Materials from Mesh property and select the physics material from the Mesh Surfaces - Entire object property list to assign it.

Warning

There is a known issue where switching physics material libraries between colliders such as Ragdoll PhysX components will not update the physics material list to the newly assigned physics material library. To update the physics material list, click on another collider, then back on the initial collider.

Assign physics materials per face

Static entities that have PhysX collider triangle mesh assets can have physics materials assigned per face. You define physics material placement by assigning placeholder materials to the faces of the triangle collision mesh in a content creation application. The names of placeholder materials become physics material properties of the collider component.

For more information on creating PhysX collider mesh assets, see FBX Settings PhysX export (p. 439).

Note

You can use the names of the placeholder materials to automate material assignment in Lumberyard by using an _ in the placeholder material name. The FBX exporter
parses placeholder material names found on PhysX collider mesh assets as PropertyName_PhysicsMaterialName.

- A placeholder material named Road creates a physics material property named Road on the collider component under Mesh Surfaces.
- A placeholder material named Road_Dirt creates a physics material property named Road and attempts to assign the physics material named Dirt to the Road property, assuming that the assigned physics material library contains a physics material named Dirt.

In the example below, the PhysX collider mesh asset has two placeholder materials, Mud and Grass. The FBX exporter creates physics material properties using the placeholder material names, and corresponding physics materials are assigned from the physics material library.

**Terrain physics materials**

Terrain is assigned the default physics material. Use the Terrain Texture Layers (p. 1232) editor to assign materials to different surface types.
Lumberyard User Guide

PhysX Scene Queries

Note

- To add terrain physics materials, you must have a PhysX Terrain (p. 767) component in your scene.
- To ensure that the engine generates a unique surface ID that can be assigned with the physics material, you must change the layer's material. If multiple layers share the same surface ID, they must have the same physics material assigned.

PhysX Scene Queries

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use physics raycast and shape cast queries to determine whether a specific line segment intersects physics geometry. For example, you might want to determine what object is in front of another object, or test a line of sight. For a shape cast, the line segment is in the form of a desired
shape (for example, a sphere). All scene queries are performed on a `Physics::World` object. For more information, see PhysX World Programming Notes (p. 2796).

You can use scene queries to find nearby objects using the following methods.

**Topics**
- Raycast (p. 2805)
- Shapecast (p. 2808)
- Overlap (p. 2810)

**Note**
Scene queries can have a performance cost.

**Raycast**

Raycast queries are the most common scene query. A raycast query takes a point and direction, with a distance, and returns the closest collider that intersected the ray.

**Example**

The raycast query intersects the pentagon only.

To perform a raycast query, use the `WorldRequestBus`.

**Example RayCast Closest**

```cpp
Physics::RayCastRequest request;
Physics::RayCastHit result;
WorldRequestBus::BroadcastResult(result, &WorldRequests::RayCast, request);
```

You can also retrieve objects that intersect with the ray by using the multiple version.

**Example RayCast Multiple**

```cpp
Physics::RayCastRequest request;
vector<Physics::RayCastHit> results;
WorldRequestBus::BroadcastResult(result, &WorldRequests::RayCastMultiple, request);
```
**Note**
You can specify the maximum number of hits that can be collected from a RayCastMultiple query. You can specify the **RayCast Buffer Size** property in the **PhysX Configuration** window. For more information, see World Configuration (p. 2775).

The following tables describe the properties for the RayCastRequest and RayCastHit objects.

### RayCastRequest Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m_distance</strong></td>
<td>Maximum distance along the ray to test for intersections.</td>
</tr>
<tr>
<td><strong>m_start</strong></td>
<td>World space point where the ray starts.</td>
</tr>
<tr>
<td><strong>m_direction</strong></td>
<td>Direction to cast the ray. This vector must be normalised.</td>
</tr>
<tr>
<td><strong>m_collisionGroup</strong></td>
<td>Specifies which layers to test against. Use this to test only against specific layers.</td>
</tr>
<tr>
<td><strong>m_filterCallback</strong></td>
<td>Custom callback function provided</td>
</tr>
</tbody>
</table>
**Raycast**

*Property*

**m_queryType**

Include either static, dynamic objects, or both.

---

**RayCastHit Properties**

*Property*

**m_distance**

Distance along the ray at which the hit was found.

**m_position**

Position in world space of the hit.

**m_normal**

Normal in world space of the hit surface.

**m_body**

Body that was hit.
Shapecast

A shapecast query is similar to a raycast query except that a shapecast query takes a shape as well as a point and direction. The shape is swept along the ray to form a volume. Anything that intersects with this volume is returned from the query.

Example

The shapecast query is in the shape of a sphere and intersects with the rectangle and pentagon entities.

Example ShapeCast Closest

```cpp
Physics::ShapeCastRequest request;
Physics::RayCastHit result;
WorldRequestBus::BroadcastResult(result, &WorldRequests::ShapeCast, request);
```

Similar to raycasting, there is a multiple version for retrieving all objects that intersect with the volume.
Example ShapeCast Multiple

```cpp
Physics::ShapeCastRequest request;
vector<Physics::RayCastHit> results;
WorldRequestBus::BroadcastResult(results, &WorldRequests::ShapeCastMultiple, request);
```

**Note**
You can specify the maximum number of hits that can be collected from a ShapeCastMultiple query. You can specify the **Shapecast Buffer Size** value in the **PhysX Configuration** window. For more information, see World Configuration (p. 2775).

ShapeCastRequest Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_distance</td>
<td>Maximum distance along m_direction to test.</td>
</tr>
<tr>
<td>m_start</td>
<td>Transform in world space where the shape cast begins.</td>
</tr>
<tr>
<td>m_direction</td>
<td>Direction to cast. The vector must be normalised.</td>
</tr>
<tr>
<td>m_shapeConfiguration</td>
<td>Shape that should be swept along the ray.</td>
</tr>
<tr>
<td>m_collisionGroup</td>
<td>Specifies which layers to test against. Use</td>
</tr>
</tbody>
</table>
**Overlap**

Overlap queries are simpler, as they don’t take a direction or distance. Overlap queries simply return all objects that intersect a shape at specified location in the world. There is only one multiple version using this method.

**Example**

The overlap query is a sphere shape that intersects with both entities.
Example Overlap Closest

```cpp
Physics::OverlapRequest request;
vector<Physics::OverlapHit> results;
WorldRequestBus::BroadcastResult(results, &WorldRequests::Overlap, request);
```

**Note**
You can specify the maximum number of hits that can be collected from an overlap query. You can specify the Overlap Query Buffer Size value in the PhysX Configuration window. For more information, see World Configuration (p. 2775).

OverlapRequest Properties

- **m_pose**: Transform in world space of the shape.
- **m_shapeConfiguration**: Shape to use for the overlap.
- **m_collisionGroup**: Specifies which layers to
Overlap

Property

Description

test against. Use this to test only against specific layers.

m_filterCallback

Custom callback function provided by the same to filter out specific entities.

m_queryType

Includes static, dynamic, or both.

OverlapHit Properties

Property

Description

m_body

Body that was hit.

m_shape

Shape on the body that was hit.

m_material

Material on the shape that was hit.
Dynamic joints with PhysX

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

PhysX joint components constrain the position and orientation of one PhysX rigid body called the **follower**, relative to another PhysX rigid body, called the **leader**. The leader rigid body will have rotational freedom in zero, one, or two axes around the joint, depending on the type of PhysX joint.

The example image below is a simple demonstration of the three joint types. In each example, the blue sphere is the follower rigid body. The joints are centered on their respective follower rigid bodies. The red sphere is the leader rigid body. For clarity, the ball joint and hinge joint follower rigid bodies are set to fixed positions, but they can be dynamic rigid bodies like the fixed joint example. Also note that the joints can be offset from the follower rigid body using the **Local Position** and **Local Rotation** properties of the joint component.

Contents
- PhysX joint types (p. 2814)
- PhysX joint setup (p. 2814)
- PhysX Joint configuration (p. 2815)
  - Position mode (p. 2815)
  - Rotation mode (p. 2815)
  - Snap position mode (p. 2816)
- Snap rotation mode (p. 2817)
- Maximum Force and Maximum Torque modes (p. 2818)
- Swing limits mode (p. 2819)
- Twist limits mode (p. 2820)
- Stiffness and Damping modes (p. 2821)
- Notes on stability (p. 2822)

PhysX joint types

See the linked component reference below for information on the three PhysX joint types:

- PhysX Ball Joint component reference (p. 726) - The PhysX Ball Joint component allows freedom of rotation of the leader rigid body in two axes.
- PhysX Fixed Joint component reference (p. 745) - The PhysX Fixed Joint component does not allow freedom of rotation of the leader rigid body in any axis.
- PhysX Hinge Joint component reference (p. 755) - The PhysX Hinge Joint component allows freedom of rotation of the leader rigid body in one axis.

PhysX joint setup

The setup for each joint type is the same.

To set up a PhysX joint

1. Create an entity for the joint and the follower rigid body.
   a. Create a new entity. Right click in Perspective and choose Create entity from the context menu.
   b. Add a PhysX Rigid Body component to the entity.
   c. Add a PhysX Collider component to the entity. This is required for angle limits to work correctly. Joints still work without a PhysX Collider component but angle limits and might not be enforced. This is also true when using trigger colliders.
   d. Add one of the PhysX joint components:
      - PhysX Ball Joint
      - PhysX Fixed Joint
      - PhysX Hinge Joint
   e. The joint's position and orientation are expressed as an offset from the follower. Adjust the position and orientation of the joint by changing the Local Position and Local Rotation fields in the PhysX joint component.

2. Create an entity for the leader rigid body.
   a. Create a new entity.
   b. Add a PhysX Rigid Body component to the entity.
   c. Add a PhysX Collider component to the entity.

3. Assign the leader entity to the Lead Entity property of the PhysX joint.
   a. In the Joint component, click the Target button to the right of the Lead Entity property to start entity selection.
   b. In Perspective, click on the leader entity to select it and assign it to the Lead Entity property.
PhysX Joint configuration

Joint components have an **Edit** button that enables component edit mode. In component edit mode, you can edit the properties of the joint in **Perspective**. You can use one of several edit contexts in component edit mode. Press the **Tab** key to cycle through the edit mode contexts. The current context is displayed at the bottom of the **Perspective** pane.

**Position mode**

**Applies to:** All Joints

Position mode displays a translate gizmo that you can click and drag to adjust the **Local Position** of the joint relative to the entity transform.

**Rotation mode**

**Applies to:** All Joints
Rotation mode displays a rotation gizmo that you can click and drag on any axis to adjust the Local Rotation of the joint relative to the entity transform.

**Snap position mode**

**Applies to:** Ball Joint and Hinge Joint
Snap position mode displays a highlight bounding box and target when you hover over an entity. Click the entity to snap the joint **Local Position** to the highlighted entity's position. If **Select Lead on Snap** is enabled in the joint properties, the entity will be assigned to the joint's **Lead Entity** property. Any entity can be selected except the follower entity.

**Snap rotation mode**

**Applies to:** Ball Joint
Snap rotation mode displays a highlight bounding box and target when you hover over an entity. Click the entity to snap the joint **Local Rotation** to the highlighted entity's rotation. Any entity can be selected except the leader entity.

**Maximum Force and Maximum Torque modes**

**Applies to:** All Joints
Maximum force and maximum torque modes display a gray box that you can click and drag to adjust the **Maximum Force** and **Maximum Torque** properties. The maximum force and maximum torque modes and properties are available only when the **Breakable** property is enabled for the joint.

### Swing limits mode

**Applies to:** Ball Joint
Swing limits mode displays a ring gizmo at the local root of the joint that you can use to rotate the swing limits on the joint's x-axis, and a scale gizmo that you can use to scale the swing limits uniformly or non-uniformly on the joint's y- and z-axes. Swing limits mode is available only when the Limits property is enabled for the ball joint component.

**Twist limits mode**

**Applies to:** Hinge Joint
Twist limits mode displays two ring gizmos that you can click and drag to adjust the **Positive angular limit** and **Negative angular limit** properties. The red ring adjusts the positive limit and the green ring adjusts the negative limit. Twist limits mode is available only when the **Limits** property is enabled for the hinge joint component.

**Stiffness and Damping modes**

**Applies to:** Ball Joint and Hinge Joint
Stiffness and damping modes display a gray box that you can click and drag to adjust the Stiffness and Damping properties. The stiffness and damping modes and properties are available only when the Soft limit property is enabled for the joint.

Notes on stability

The iterative solver used by PhysX joints may not be able to maintain constraints in some configurations. For example, the solver might fail to converge. The result is unstable or unexpected motion during simulation. The PhysX documentation describes configurations that could help avoid such occurrences. Please see Configuring Joints for Best Behavior in NVIDIA’s PhysX Joint documentation.

Create global or localized wind forces with PhysX

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create global wind forces or wind forces contained within a collider volume with the PhysX Force Region component. Wind forces act on PhysX entities, such as Cloth, that can be affected by wind.
1. Create an entity for the wind provider.

2. Add a Tag component to the entity. The Tag component will specify whether the wind force is global or localized to a collider volume.

3. Specify whether the wind provider will be global or localized. To determine what value to use for the tag component, from the File menu, choose PhysX Configuration. At the bottom of the Global Configuration tab there is a section labeled Wind Configuration.

<table>
<thead>
<tr>
<th>Wind Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global wind tag</td>
</tr>
<tr>
<td>Local wind tag</td>
</tr>
</tbody>
</table>

Wind Configuration has properties for a Global wind tag and a Local wind tag. You may use the default values or set them as desired. These tags are used by the PhysX wind system to detect entities that provide wind data.

For this example use the Global wind tag property value. In the Tag component of the entity, add a tag element and give it the value global_wind.

**Note**
If you choose to use the Local wind tag, the wind will only affect entities that are inside the volume of the PhysX Collider you create in the next step.

4. Add a PhysX Collider component. If you are using the Local wind tag, this collider will define the volume of the wind force. With the Global wind tag, the size and position of this collider does not matter because the wind will be global, however, the collider will provide a useful visualization for the global wind force.

5. Set the PhysX Collider component’s Shape property to Box.

6. Scale and position the PhysX Collider component. Use the move tool to place the entity in the level, and set the Box Dimensions property in the PhysX Collider component as desired.

7. Add a PhysX Force Region component to the entity. The PhysX Force Region component creates the wind force.

8. In the PhysX Force Region component, specify 10.0 in the Direction property’s Y component to set a direction for the wind force.

9. In the PhysX Force Region component, specify 50.0 in the Magnitude property to create magnitude for the wind force.

The PhysX collider box in Perspective displays cones and vectors representative of the wind force direction.
10. To test the wind provider, add an entity with a **Cloth** mesh. There is an example slice asset you can drag into **Perspective** from **Asset Browser**. Navigate to **Gems\NvCloth\Assets\slices** in **Asset Browser**. Locate the `cloth_locked_corners_two.slice` and drag and drop the slice into **Perspective**.

11. Use the **Move** tool to place the cloth entity. If you are using the **Local wind tag**, you must place the cloth asset inside the **PhysX Collider** volume.
12. Press **Control + G** to test the wind simulation.
Debugging PhysX

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The PhysX system has the following features that you can use to debug issues.

**Note**
You must first enable the **PhysX Debug** (p. 1188) gem.

**Topics**
- PhysX Debug Console Variables (p. 2826)
- Debugging with the ImGui Tool (p. 2827)
- Debug Options in the PhysX Configuration (p. 2832)

**PhysX Debug Console Variables**

Enter the following console variables to debug your PhysX issues.

Sets your preferences for debugging. As a recommended best practice, enter this console variable command as your first step for debugging.

**Example**

```
physx_Debug 1
```

You can specify the following values:

- **1** – Enable debug visualizations. By default, this value enables the collision shapes and edges for your PhysX entities.
• 2 – Enables all configuration options. This enables all the available visualization options.
• 3 – Toggles the proximity based collider visualization. This value applies only to mesh colliders. See Physics asset colliders (p. 739).
• 0 – Disables debug visualizations.

Toggles a visual culling box frame.

**Example**

```plaintext
physx_CullingBox 1
```

Adjusts the culling box size to **100**. Enter 0 to disable culling.

```plaintext
physx_CullingBoxSize 100
```

Connects to the PhysX Visual Debugger. You must have the PhysX Visual Debugger open to run this command. See Debugger Configuration (p. 2790).

```plaintext
physx_PvdConnect
```

Disconnects from the PhysX Visual Debugger. You must have the PhysX Visual Debugger open to run this command. See Debugger Configuration (p. 2790).

```plaintext
physx_PvdDisconnect
```

For more information, see Using the Console Window (p. 210).

**Debugging with the ImGui Tool**

In game mode, you can configure the PhysX debug settings using the immediate mode graphical user interface (ImGui) tool.

**Note**
You must enable the ImGui gem to access this tool. For more information, see Enabling Gems (p. 1064).

**To debug with the ImGui tool**

1. Press **Ctrl+G** to enter gameplay mode.
2. Press the **Home** key to open the **ImGui** tool. The **PhysX Debug** menu appears under the **Perspective** viewport.
3. Click **PhysX Debug**.
Example

4. You can make the following changes.

<table>
<thead>
<tr>
<th>PhysX Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug visualizations</td>
<td>Enables debug visualizations mode.</td>
</tr>
<tr>
<td>PhysX Color Mappings</td>
<td></td>
</tr>
<tr>
<td>Visualize Colliders</td>
<td>Enables colliders to appear.</td>
</tr>
<tr>
<td>Culling</td>
<td></td>
</tr>
<tr>
<td>Collisions</td>
<td></td>
</tr>
<tr>
<td>Body</td>
<td></td>
</tr>
<tr>
<td>Contact</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td></td>
</tr>
</tbody>
</table>

This is the same as the `physx_Debug` console variable.

<table>
<thead>
<tr>
<th>PhysX Debug</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics Debug</td>
<td></td>
</tr>
<tr>
<td>Disable all settings</td>
<td></td>
</tr>
<tr>
<td>PhysX Scale</td>
<td></td>
</tr>
</tbody>
</table>

Version 1.28
2828
Debugging with the ImGui Tool

### PhysX Debug Settings

- **console variable.**

### Culling

can specify the following options:

- **Wireframe**
  - Displays wireframes in the viewport.
- **Size**
  - Click and drag the slider to specify the size of the wireframes.
  
  As a best practice, keep this value small to prevent performance issues.
**Debugging with the ImGui Tool**

### Debug Settings

#### Collisions

Enables debugging for collision types. You can specify the following options:

- Shapes
- Edges
- Normals
- Aabbs
- Axis
- Compounds
- Static
- Dynamic

#### Bodies

Enables debugging for body types. You can specify the following options:

- Axes
- Mass Axes
- Linear Velocity
- Angular Velocity
**Contact**

deb ginning for contact types.
You can specify the following options:

- Point
- Normal

**Character**

deb ginning for characters.
You can specify the following options:

- Joint Limits
- Mbp Regions
- Actor Axes

**PhysX Color Mappings**

color mappings.
### Debug Options in the PhysX Configuration

You can also specify debug settings in the **PhysX Configuration** tool. See [Debugger Configuration](#) (p. 2790).

### PhysX Best Practices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the [AWS Game Tech blog](#) to learn more.

See the following best practices when working with PhysX.

- Colliders intersecting with terrain can result in unexpected behavior. For example, the object might rocket into space, jitter, or slow down performance. Avoid intersecting colliders with terrain. If you need to intersect a collider with terrain, use a small value for the collider size. These scenarios can be...
mitigated by clearing the Persistent Contact Manifold check box in the Global Configuration tab in the PhysX Configuration tool.

- The PhysX Character Controller component must be on the same entity as the Actor component in order to work with the Animation Editor.

- If you select the Static check box in the Transform component for an entity that also has a PhysX Rigid Body component, the rigid body behaves statically and a warning appears about the incompatibility of the PhysX Rigid Body component and the Static transform option.

- When adding the PhysX Collider component to entities, prefer a primitive shape (box, capsule, or sphere) for the collider. These shape colliders offer better performance and should be used when possible.

**Warning**
Avoid non-uniform scaling on entities that have PhysX components.
If you must apply a non-uniform scale to an entity that has PhysX components, do the following:
1. Create a child entity under the entity with the PhysX components.
2. On the child entity, add a Mesh component.
3. Ensure that the Transform component uses non-uniform scaling.
4. Update the PhysX Collider dimensions as needed.
5. Update the center of mass so that it is the center of the object.

## Simulated destruction with NVIDIA Blast

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With NVIDIA Blast in Amazon Lumberyard, you can simulate destruction by authoring blast assets in SideFX Houdini and creating entities with the Blast Family and Blast Family Mesh Data components.

To use NVIDIA Blast, you must enable the NVIDIA Blast gem (p. 1181).

**Note**
NVIDIA Blast for Lumberyard requires a SideFX Houdini commercial or indie license to create assets. The apprentice license is not sufficient. For more information on Houdini, see SideFX's home page.
The precompiled Houdini plug-ins supplied with the NVIDIA Blast gem require Houdini 18.0.

## NVIDIA Blast features

These are the features that NVIDIA Blast provides:

- Fracture geometry and author blast assets in SideFX Houdini with the provided Houdini plug-ins and Houdini Digital Assets (HDAs).
- Create multiple levels of destruction for simulation.
- Automatically process and quickly set up assets with the provided Python Asset Builder for NVIDIA Blast.
- Create blast materials to define the forces required to trigger destruction simulation on blast assets.
Using NVIDIA Blast

See the following topics for detailed information about NVIDIA Blast.

- **Blast Family component (p. 564)** - Blast Family component reference.
- **Blast Family Mesh Data component (p. 566)** - Blast Family Mesh Data component reference.
- **Install NVIDIA Blast plug-ins (p. 2835)** - Install the NVIDIA Blast plug-ins and Houdini Digital Assets.
- **Create assets for NVIDIA Blast (p. 2836)** - Fracture meshes in Houdini and export assets for NVIDIA Blast.
- **Processing assets for NVIDIA Blast (p. 2842)** - Process NVIDIA Blast assets for Lumberyard.
- **Simulate destruction with NVIDIA Blast (p. 2845)** - Create entities with NVIDIA Blast assets and simulate destruction.
- **Partial destruction with NVIDIA Blast (p. 2849)** - Use attributes to create partial destruction.
- **Specify destruction properties with Blast materials (p. 2851)** - Use a blast material to define the force required to trigger destruction.
- **NVIDIA Blast visual debugger (p. 2854)** - Use the visual debugger for NVIDIA Blast.
- **Script Canvas nodes for NVIDIA Blast (p. 2856)** - Script destruction simulation in Script Canvas.

**NVIDIA Blast references**

NVIDIA Blast documentation at the NVIDIA GAMEWORKS developer portal.

Install SideFX Houdini plug-ins for NVIDIA Blast

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To create NVIDIA Blast assets, you must install a set of plug-ins and Houdini Digital Assets for SideFX Houdini.

**Note**
NVIDIA Blast for Lumberyard requires a SideFX Houdini commercial or indie license to create assets. The apprentice license is not sufficient. For more information on Houdini, see SideFX's home page.

The precompiled Houdini plug-ins supplied with the NVIDIA Blast gem require Houdini 18.0.
Install NVIDIA Blast plug-ins

To install plug-ins

1. Run `install_plugin.bat` from the `houdini` directory located at `lumberyard_installation\dev\Gems\Blast\houdini`.
2. Grant the installer script administrator privileges when requested.
   
   The plug-ins are installed into various directories in `C:\Users\user_name\Documents\houdini18.0`.
3. To verify the installation, you can check for the following files in your Houdini user directory.
   
   The `C:\Users\user_name\Documents\houdini18.0\otls` directory contains these three files:
   - `blastExport.hda`
   - `fractureHierarchy.hda`
   - `fractureSingle.hda`

   The `C:\Users\user_name\Documents\houdini18.0\toolbar` directory contains these three files:
   - `default.shelf`
   - `Fracture.shelf`
   - `shelf_tool_assets.json`

   The `C:\Users\user_name\Documents\houdini18.0\dso` directory contains these five files:
   - `BlastExportPlugin.dll`
   - `BlastExportPlugin.exp`
   - `BlastExportPlugin.ilk`
   - `BlastExportPlugin.lib`
   - `BlastExportPlugin.pdb`

   The `C:\Users\user_name\Documents\houdini18.0\dependencies` directory contains eleven `NvBlast*\.dll` files and four `PhysX*\.dll` files.
4. Verify that the installation script has added your `houdini` user directory to the `PATH` environment variable in Windows:

   `C:\Users\user_name\Documents\houdini18.0`
Enable the Blast tool shelf

The NVIDIA Blast installation for Houdini includes a tool shelf that you can enable to speed up the process of creating assets for NVIDIA Blast.

To enable the Blast tool shelf

1. In Houdini, choose the + button in the Toolbar to the far right.
2. From the list, choose Shelves to expose the Shelf list.
3. From the Shelf list choose Fracture tools for Blast to add the shelf to the Toolbar.

Create assets for NVIDIA Blast

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Assets for NVIDIA Blast are created in Houdini with the provided tools and plug-ins. Follow the steps below to create and export blast assets for Lumberyard.

Note

These steps can be applied to any mesh. For best results, the mesh should be fully enclosed. Custom normals, UV sets, and vertex color streams can be applied to the root mesh and the resulting chunks.

Contents

- Fracture a mesh for NVIDIA Blast (p. 2837)
- Export an asset for NVIDIA Blast (p. 2841)
Fracture a mesh for NVIDIA Blast

To fracture a mesh

1. Create an object in Houdini containing a mesh. To avoid potential warnings and errors when runtime assets are generated, ensure you add UV and Normal attributes to the mesh.
2. At the SOP level, append a Name SOP to the geometry to be fractured.
3. In the Name SOP, enter root for the Name parameter. This applies a name primitive attribute to the geometry with the value root so that the root mesh can be identified.
4. You can use the Fracture Selection tool to automatically set up a fracture network. Before you continue, it's important to understand the two Houdini Digital Assets provided by the NVIDIA Blast gem to fracture geometry, Fracture Single and Fracture Hierarchy:
   - Fracture Single fractures the provided geometry into a specified number of chunks. Multiple levels of fracture can be created by appending a Fracture Single SOP to already fractured geometry.
   - Fracture Hierarchy fractures the provided geometry, and fractures the resulting chunks for each additional level specified by the Fracture levels parameter. If you specify 2 Fracture levels and 10
Fractures per level, the root geometry is fractured into ten chunks and each chunk is fractured into ten chunks.

Warning
The number of chunks generated by Fracture Hierarchy grows exponentially. Setting the Fracture levels and Fractures per level too high can generate assets that are too large to load and simulate with NVIDIA Blast. For fine control over multiple fracture levels, use multiple Fracture Single SOPs.

Note
Both Fracture Single and Fracture Hierarchy Houdini Digital Assets (HDAs) are created with standard Houdini SOPs. You can open these HDAs to view their network contents and use them as the basis for custom fracture SOPs for your projects.

The provided Fracture Selection tool adds a Fracture Single SOP and feeds the root mesh and the resulting chunks from the fracture operation into a Merge SOP.

With the Name SOP selected, choose Fracture Selection from the Fracture tools for Blast shelf. A Fracture Single SOP is appended to the network and input into a Merge SOP along with the root mesh. Note that the Fracture Single SOP specifies root as the Group to fracture and chunk as the Chunk name prefix.

5. To visualize the fracture:
   a. Ensure the Display flag is enabled on the Merge SOP.
   b. Choose the Display groups and attribute list toggle from the Display options toolbar to the right of the Perspective viewport. This enables the groups and attributes overlay in the Perspective viewport.
   c. In the groups and attributes overlay, choose the Gear button to open the list options.
d. In the list options, choose Attributes and select name to visualize the geometry by its name attribute. You can see the hierarchy of geometry in the list, and the mesh chunks are colored by their name primitive attribute in the perspective viewport.

Note
With Houdini’s geometry select mode set to primitive, you can press the S key to enter select mode and click on any chunk, at any level, in the groups and attributes overlay, to select that chunk and its descendants.

6. It might be difficult to see where the mesh is fractured if the root geometry is complex. You can use Exploded View to better inspect the fracture result. To see an exploded view of the fractured mesh:

   a. Append a Partition SOP to the Merge SOP.
   b. Append an Exploded View SOP to the Partition SOP.
   c. Enable the Display flag on the Exploded View SOP.
7. Fractures can be hierarchical, that is, a fracture chunk can be further fractured. To add an additional fracture level:
   
a. Add a Fracture Single SOP to the network.

b. Connect the output of the Fracture_root SOP to the input of the newly added Fracture Single SOP.

c. Connect the output of the newly added Fracture Single SOP to the input of the Merge SOP.

d. In the Fracture Single SOP, specify a chunk to fracture. In the example below, chunk1 (the rabbit's face) is specified in the Group to fracture parameter using the path root/chunk1.
Export an asset for NVIDIA Blast

With the mesh fractured, you must export .fbx and .blast assets to be processed by Asset Processor for use in Amazon Lumberyard.

To export an asset

1. Add a Blast Export SOP to the network.
2. Connect the output of the Merge SOP to the input of the Blast Export SOP.
3. Ensure the Root Name and Chunk name prefix parameters are properly set.
4. Set the Object name parameter to the desired name. The exported .fbx and .blast files are named by this parameter’s value.
5. Set the Output Directory parameter. By default, the .fbx and .blast files are exported to the directory where the Houdini scene file is saved.
6. Disable the Static root parameter.

- If the Static root parameter is enabled, the root mesh is a static rigid body and unaffected by gravity. Sufficient force fractures the root mesh and the chunks are dynamically simulated.
- If the Static root parameter is disabled, the root mesh is dynamic, treated like a rigid body, and affected by gravity. Sufficient force fractures the root mesh and the chunks are dynamically simulated.

Note
All chunks are simulated as dynamic rigid bodies. You can make non-root chunks into static rigid bodies by adding the word static to the name primitive attribute of the non-root
chunk. This is useful for scenarios where you want to destroy part of an entity while leaving part of the entity in place. For more information, see Partial destruction with NVIDIA Blast (p. 2849).

7. Choose **Export Blast Asset** to generate a `.blast` asset.

8. Choose **Export FBX** to generate a `.fbx` asset.

### Processing assets for NVIDIA Blast

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the **AWS Game Tech blog** to learn more.

Blast assets that are exported from Houdini must be processed by **Asset Processor** to generate run-time assets. There are two methods for processing Blast assets for use in Lumberyard: an automated method using Python Asset Builder, and a manual method where you select the meshes to be processed.

**Important**

Processing meshes automatically is less time consuming than manually setting up blast meshes to process. It is important to understand both processes, however. You might use the automatic process for the initial export and then manually edit the mesh assets to add modifiers for custom normals or vertex color streams.
Once you have manually edited the .fbx asset in FBX Settings, a .assetinfo file is created for the .fbx asset. The .assetinfo file prevents the mesh assets from being processed automatically by Python Asset Builder. Any further changes made to the source .fbx file, such as adding or removing fracture levels or chunks, must be manually edited in FBX Settings.

Contents

- Process Blast assets automatically (p. 2843)
- Process Blast mesh assets manually (p. 2844)

Process Blast assets automatically

When blast assets are processed automatically, a blast slice asset is created that adds the blast mesh chunks to the Blast Family Mesh Data component. If you have a blast asset with dozens of chunks, automatic processing with the Python asset builder for NVIDIA Blast can save some time.

**Note**
Automatically processing assets for NVIDIA Blast requires that your project has been built with the Python Asset Builder and EditorPythonBidnings gems enabled. For more information, see Python Asset Builder gem (p. 1192).

To process Blast assets automatically

1. Copy the .blast and .fbx files for your Blast asset into an asset directory in your project.
2. Start Lumberyard Editor. Asset Processor detects the .blast and .fbx files and generates the runtime mesh assets, a blast asset, and a blast slice asset.
3. You can verify that the assets have processed successfully in the Jobs tab of Asset Processor. If you need to reprocess the assets, do the following:
   a. In Asset Processor, select the Assets tab.
   b. Right-click the asset in the asset list to open the context menu.
   c. Choose Reprocess File from the context menu.
Process Blast mesh assets manually

Manually processing blast assets requires you to add a mesh group for each chunk mesh so that Asset Processor can generate the run-time asset. Manual processing also requires you to add each run-time mesh to the Blast Family Mesh Data component. If you need to add modifiers to the blast mesh chunks, such as specifying a vertex color stream, you must use this manual process.

To process Blast assets manually

1. Copy the .blast and .fbx files for your Blast asset into an asset directory in your project.
2. Start Lumberyard Editor.
3. Locate the .fbx asset in Asset Browser and double-click on the asset to open FBX Settings.
4. Choose the Meshes tab.
5. Create one Mesh group for each mesh in the .fbx asset.
   a. If there is an existing Mesh group, ensure it has only the first mesh selected from the Select Meshes list.
   b. Add a new Mesh group by choosing Add another mesh.
   c. Add the next mesh to the new Mesh group by choosing the Hierarchy button to the right of Select meshes and selecting the next mesh from the list.
   d. Repeat steps b and c until each mesh in the mesh list is assigned to its own Mesh group.
6. **Optional:** If the meshes require special processing, such as a vertex color stream provided by the *Mesh (Advanced)* modifier, add modifiers to each mesh group as required.

7. Choose the **Update** button in the bottom right of *FBX Settings*. *Asset Processor* generates the run-time assets. A **File progress** window appears and displays feedback about the process.

**Simulate destruction with NVIDIA Blast**

This feature is an **experimental** release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the **AWS Game Tech blog** to learn more.
To use NVIDIA Blast assets in Lumberyard, create an entity, add a **Blast Family** component, add a **Blast Family Mesh Data** component, and then assign the blast assets to the components.

**Note**

To quickly test NVIDIA Blast simulation, the following steps assume that the assets have been exported from Houdini with **Static root** disabled in the **Blast Export** SOP. With **Static root** disabled, the NVIDIA Blast asset is dynamic, and destruction can be triggered by dropping the entity on a PhysX collision surface such as **PhysX Terrain**. If **Static root** is enabled, the root asset is static, and destruction must be triggered by an external force, such as a projectile impact. For more information, see Create assets for NVIDIA Blast (p. 2836).

**Contents**

- Create an entity for NVIDIA Blast (p. 2846)
- Add automatically processed mesh assets to a NVIDIA Blast entity (p. 2847)
- Add manually created mesh assets to a NVIDIA Blast entity (p. 2847)
- Test NVIDIA Blast destruction simulation (p. 2848)

### Create an entity for NVIDIA Blast

When you create an entity, you add the NVIDIA Blast functionality and define how the asset destructs.

**To create an entity for NVIDIA Blast**

1. Ensure that the terrain has a **PhysX Terrain** level component. In **Level inspector**, choose **Add Component** and select **PhysX Terrain** from the component list.
2. Create a new entity. Right-click in **Perspective** and choose **Create entity** from the context menu.
3. Add a **Blast Family** component to the entity. In **Entity Inspector**, choose **Add Component** and select **Blast Family** from the component list. The **Blast Family** component adds NVIDIA Blast functionality to the entity. For more information, see **Blast Family component** (p. 564).
4. Set the **Blast asset** for the **Blast Family** component. Click the **Folder** button to the right of the **Blast asset** property and choose the `.blast` asset in the Blast Asset selection window.
5. Set the **Blast Material** for the **Blast Family** component. Blast materials define how much damage various forces cause to the bonds holding the fractured asset together, and how much damage is required to cause destruction. For more information see Specify destruction properties with Blast materials (p. 2851).

6. Add a **Blast Family Mesh Data** component to the entity. In Entity Inspector, Choose Add Component and select **Blast Family Mesh Data** from the component list. The **Blast Family Mesh Data** component adds NVIDIA Blast meshes to the entity. For more information, see **Blast Family Mesh Data component** (p. 566).

If you have processed your mesh assets with **Python Asset Builder**, follow the steps in the section: Add automatically processed mesh assets to a NVIDIA Blast entity (p. 2847).

If you have manually edited your mesh assets with **FBX Settings**, follow the steps in the section: Add manually created mesh assets to a NVIDIA Blast entity (p. 2847).

**Add automatically processed mesh assets to a NVIDIA Blast entity**

**Python Asset Builder** creates a blast_slice asset when it processes your NVIDIA Blast assets. The blast slice automatically adds the mesh assets and material to the **Blast Family Mesh Data** component.

**To add automatically processed mesh assets**

1. In the **Blast Family Mesh Data** component, set a material for the Blast mesh. Click the **Folder** button to the right of the **Material** property and select a material from the material selection window.

2. In the **Blast Family Mesh Data** component, set the **Blast Slice** property. Click the **Folder** button to the right of the **Blast Slice** property and select the asset from the Blast Slice selection window.

Enable the **Show mesh assets** property if you would like to view the mesh list.

3. The entity is now set up to simulate destruction. Continue to the section: Test NVIDIA Blast destruction simulation (p. 2848).

**Add manually created mesh assets to a NVIDIA Blast entity**

If your NVIDIA Blast mesh assets have been manually edited in **FBX Settings**, use the following steps to add the mesh assets to the entity.
To add manually processed assets

1. In the **Blast Family Mesh Data** component, set a material for the Blast mesh. Click the **Folder** button to the right of the **Material** property and select a material from the material selection window.

2. In the **Blast family mesh data** component, enable the **Show mesh assets** property to show the mesh asset list.

3. Add a mesh slot to the list. Choose the + button to the right of **Mesh assets** to add a mesh slot to the **Mesh assets** list.

4. Add a mesh to the list. Click the **Folder** button to the right of the numbered mesh slot property and select a mesh asset from the **Static Mesh** selection window.

5. Repeat steps 3 and 4 until all of the meshes for the blast asset have been added to the **Blast family mesh data** component.

6. The entity is now set up to simulate destruction. Continue to the section: Test NVIDIA Blast destruction simulation (p. 2848).

Test NVIDIA Blast destruction simulation

Because the blast asset has been exported from Houdini with **Static root** disabled, and a **PhysX Terrain** level component has been added to the level, destruction can be tested by dropping the object on the terrain.

To test the destruction simulation

1. With the entity selected, press the Z key to enable the move tool.

2. Click and drag on the Z axis of the move gizmo to move the entity several units above the terrain.

3. Press Control + P to view simulation. The entity drops and shatters when it collides with the terrain.

4. Press Control + P to end the simulation.
Partial destruction with NVIDIA Blast

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In some scenarios, you might want to partly destroy an entity. For example, you create a destructible wall, but want the bottom of the wall to remain in place as a static mesh with colliders after the top of the wall takes damage from a projectile and is destroyed. You can achieve this by adding static to the name primitive attribute of non-root mesh chunks in Houdini.

Make non-root chunks static in NVIDIA Blast

Use Houdini to make non-root chunks static, which results in partial destruction of the asset.
To make non-root chunks static

1. In Houdini, add an Attribute String Edit SOP to the network.
2. Wire the Attribute String Edit SOP into the network.
   a. Connect the output of the Merge SOP to the input of the Attribute String Edit SOP.
   b. Connect the output of the Attribute String Edit SOP to the input of the Blast Export SOP.
3. In the Attribute String Edit SOP, in the Attributes tab, enable the Primitives parameter, and select the name attribute from its list.
4. In the Attribute String Edit SOP, in the Filter tab, set the From parameter to the path name of a chunk you would like to make static, such as root/chunk5.
5. In the Attribute String Edit SOP, in the Filter tab, set the To parameter to the same path as above. Append static to the path; for example, root/chunk5static.

   **Important**
   If the specified chunk has been fractured, its descendants are also static when exported to Lumberyard.
6. You can add additional chunks to the Attribute String Edit SOP. Choose the + icon next to the Number of filters parameter to add a filter. Repeat steps 4 and 5 to make another chunk static.

In the image below, the chunks that comprise the back half of the rabbit have been made static. Note that 12 filters have been added to the filter list in the Attribute String Edit SOP. You can see the renamed chunks in the groups and attributes overlay in the perspective viewport.

7. Enable the Static root parameter in the Blast Export SOP before exporting the asset.

See the result simulation in Lumberyard below. A large, invisible PhysX rigid body collider is dropped on the rabbit. The front half of the rabbit is destroyed. The chunks are simulated as dynamic rigid bodies while the back of the rabbit remains in place.
Specify destruction properties with Blast materials

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The chunks of blast assets are held together by bonds. Blast materials determine what types of forces can damage the bonds, the minimum amount of force that can damage the bonds, and how much damage the bonds can sustain before breaking.

Important
Blast materials are critical component in creating realistic destructible entities. Passive forces like gravity can have a cumulative damage effect on a blast asset, causing destruction to trigger when no obvious external force or collider has acted on the asset. Large-scale destructible assets such as buildings have large chunks that require stronger bonds and higher damage thresholds to prevent premature triggering of destruction.

Contents
• Create Blast materials (p. 2852)
• NVIDIA Blast configuration (p. 2852)
• Blast material properties (p. 2852)
Create Blast materials

Blast materials are stored in a blast material library. The blast material library for the project is set in the Blast Configuration editor. You can assign a blast material in the Blast Family component of the entity. 

To create a blast material

1. In the Tools menu, choose Asset Editor.
2. In Asset Editor, choose New from the File menu, and select Blast Material. A new Blast Material appears in the Asset Editor.
3. Set the Material name property. This property is the identifier that you use to assign the blast material to a Blast Family component.
4. Set the blast material properties. For information, see Blast material properties (p. 2852) below.
5. Additional Blast materials can be created by choosing the + button in the upper right corner of the Blast material list.
6. Save the Blast material library. In Asset Editor, in the File menu, choose Save as.
7. In the Save as... window, navigate to an appropriate directory in your project, give the blast material library a name, and choose Save.

NVIDIA Blast configuration

The blast material library for the project is set in Blast Configuration.

To use Blast Configuration

1. From the Tools menu, choose Blast Configuration.

2. Set the Blast material library for your project. Choose the Folder button to the right of Blast material library and select the blast material library that you created.
3. Set the Stress solver iterations property. Stress damage is generated by impact forces. This property sets the number of iterations that the stress solver computes per tick, for each Blast Family component. Valid values range from 0 to 50000.

You can now assign blast materials from the blast material library to Blast Family components.

Blast material properties

Blast materials function on the concepts of health and damage. Chunks are held together by bonds. Each bond has some amount of health. Damage to the bonds is dealt by impulse forces acting on the chunks. Impulse forces generate damage in two ways:

- **Vector damage** is generated by impulse forces such as a shock wave, wind, or gravity applied to the Blast entity.
• **Stress damage** is generated by impulse forces from impacts such as a projectile colliding with the Blast entity.

Material name

The name of the material. Use this identifier to select the material from the library for **Blast Family** components.

**Health**

Each bond has a default health of 1.0. The Health property is multiplied by the default bond health to set the health of all bonds in the Blast asset.

Vector damage is subtracted from Health. When Health reaches 0.0, the bond is destroyed.

Stress damage treats Health as a threshold. Damage is not applied unless the threshold is exceeded in a given tick.

**Force Divider**

Force Divider is a hardness divisor that keeps the scale of impulse forces in a reasonable range to limit stress damage.

Vector damage is given as a scalar and is not affected by Force Divider.

Stress damage calculations factor in the number of connected bonds, making the asset significantly harder to destroy. Stress damage is computed as StressDamage = ImpulseForce / (ForceDivider * ConnectedBonds).

**Minimum damage threshold**

The floor value for vector damage. If the vector damage value is less than the minimum threshold value, then the damage is ignored. If the vector damage value is greater than the minimum threshold value, then the vector damage value is checked against the **Maximum damage threshold**.

**Maximum damage threshold**

The ceiling value for vector damage. If the vector damage value is less than the maximum threshold value, then the damage is subtracted from Health. If the vector damage value is greater than the maximum threshold value, then only the value of Maximum damage threshold is subtracted from Health.

**Stress linear factor**

A factor applied to linear forces during stress damage. The default value is 1.0.
Stress angular factor

A factor applied to angular forces during stress damage. The default value is 1.0.

NVIDIA Blast visual debugger

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To use the visual debugger for NVIDIA Blast, enable the BLAST_DEBUG console variable.

To enable the variable

1. In the editor console, enter `BLAST_DEBUG 1`.
2. Click in Perspective to make the viewport active.
3. Press the F3 key to set the view to wireframe.
4. Press Control + P to run the simulation and view the debugger.
5. Press Control + P to exit the simulation.

In the below example, as the rabbit drops, the bonds between the fractured chunks are shown as green lines. On impact, the bonds change to orange as they weaken and red when they break.
Script Canvas nodes for NVIDIA Blast

This feature is an experimental release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The NVIDIA Blast gem includes several Script Canvas nodes to script destructible assets. The nodes can be found in the Blast group in Script Canvas.

Contents
- BlastFamilyComponentNotificationBus nodes (p. 2856)
- BlastFamilyComponentNotificationBus nodes (p. 2856)
- BlastFamilyDamageRequestBus nodes (p. 2856)

BlastFamilyComponentNotificationBus nodes

On Actor Created

Event notification that is invoked whenever an actor is created from a destroyed object used in the given entity.

On Actor Destroyed

Notification that is invoked whenever an actor is destroyed from a destroyed object used in the given entity.

BlastFamilyComponentNotificationBus nodes

Get Actors Data

Obtains the actor data, such as entity id and whether actor is static or not, from a destructible object used in the given entity.

BlastFamilyDamageRequestBus nodes

Capsule Damage

Full damage is dealt to all chunks and bonds that are not farther than minRadius from either position0 or position1. Linearly decreasing damage is applied if the distance is less than maxRadius.

Destroy Actor

Destroys the actors from a destroyed object used in the given entity.

Get Family Id

Get the entity id of the Blast Family in the given entity.

Impact Spread Damage

Full damage is dealt to all chunks and bonds that are not farther than minRadius from position. Decreasing damage is applied if the distance is less than maxRadius. Decreasing damage is calculated using BFS on the support graph instead of Euclidean distances.
Radial Damage

Full damage is dealt to all chunks and bonds that are not farther than \texttt{minRadius} from \texttt{position}. Linearily decreasing damage is applied if the distance is less than \texttt{maxRadius}.

Shear Damage

Deals full damage to bonds that are orthogonal to \texttt{normal}. No damage is dealt to bonds that are parallel. Damage to chunks is dependent on distance to \texttt{position}. The damage falloff between \texttt{minRadius} and \texttt{maxRadius} is linear.

Stress Damage

Stress damage is applied using \texttt{force} vector instead of a scalar value. Damage is propagated between bonds based on the support graph.

Triangle Damage

Full damage is dealt to all chunks and bonds that intersect with a triangle described by the given vertices defined by \texttt{position0}, \texttt{position1}, and \texttt{position2}.

Simulate cloth with NVIDIA Cloth

This feature is in \texttt{preview} release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

With NVIDIA Cloth in Amazon Lumberyard you can create realistic cloth simulations for entities that contain \texttt{Actor} or \texttt{Mesh} components. The NVIDIA Cloth gem provides a component you can use to simulate cloth on any mesh that has been processed with a \texttt{Cloth} modifier applied in \texttt{FBX Settings}.

To use NVIDIA Cloth you must enable the NVIDIA Cloth gem (p. 1182).

NVIDIA Cloth features

- Apply cloth data to meshes imported from .fbx files.
- Add cloth simulation to entities containing \texttt{Mesh} and \texttt{Actor} components.
- Cloth mesh simplification and static triangle removal allow you to use complex cloth meshes and yield faster cloth simulation.
- Define \texttt{Inverse mass}, \texttt{Motion constraints}, and \texttt{Backstop} per cloth particle with vertex color streams you create.
- Blend between cloth simulation and actor keyframe animation with motion constraints.
- Add cloth colliders to actors with \texttt{Animation Editor}.
- Add local wind forces to your cloth simulations or use force regions to simulate wind.
- Simulate cloth in parallel across available CPU cores, or simulate cloth on a discrete GPU that supports NVIDIA CUDA (Windows 10 only).
- NVIDIA Cloth gem's public API allows other systems and gems to access cloth simulation functionality.
• Debug cloth simulations and constraints with real-time cloth debug visualizations.

Using NVIDIA Cloth

Cloth component (p. 575) - Cloth component reference.

Cloth for Mesh components (p. 2858) - Create cloth for entities containing Mesh components.

Cloth for Actor components (p. 2861) - Create cloth for entities containing Actor components.

Per vertex properties for cloth (p. 2866) - Use per vertex properties to define the Inverse mass, Motion constraints, and Backstop to create higher quality and more predictable cloth simulations.

Cloth simulation constraints (p. 2867) - An overview of how Motion constraints and Backstop work to improve the results of cloth simulations.

Cloth visual debugger (p. 2869) - Enable the visual debugger for cloth simulations.

GPU cloth simulation with NVIDIA CUDA (p. 2871) - Install NVIDIA CUDA to enable GPU acceleration for NVIDIA Cloth.

NVIDIA Cloth references

NVIDIA Cloth documentation at the NVIDIA GAMEWORKS developer portal.

Cloth for Mesh components

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To use Cloth, you must enable the NVIDIA Cloth gem. For more information, see the NVIDIA Cloth gem (p. 1182) documentation.

You can create cloth assets for entities with Mesh components in the content creation application of your choice and import them into Lumberyard from an .fbx file. The mesh asset should have the following:

• A cloth mesh that will be simulated and rendered in Lumberyard.
  • Cloth data can be added to define per vertex mass and constraint properties using the vertex color tool in your content creation application. For more information see Per vertex properties for cloth (p. 2866).
  • Optional - Any other static mesh. For example, if you create a flag to be simulated as cloth, then you can include a mesh for the flag pole.

Note
Sample Mesh component cloth assets are located in the NVIDIA Cloth gem directory, which is located at /dev/Gems/NvCloth/Assets/Objects/cloth/Environment/.

For information on exporting mesh assets, see FBX Settings mesh export (p. 436)
Add Cloth to Mesh components

Create cloth by adding the Cloth component to an entity that has a Mesh component, and then setting the properties of the Cloth component.

1. In Lumberyard Editor, add a new entity to the level.
2. Add a Mesh component to the entity, and reference the mesh asset and material.
3. Add a Cloth component to the entity.
4. Set the cloth data of the mesh asset.
   a. Click the button beside the Mesh node property to open the FBX Settings window.

   ![FBX Settings Window]

   b. In the FBX Settings window, on the Meshes tab, and choose Add Modifier, Cloth.

   c. In the Cloth modifier area:
      i. Select the cloth mesh from the drop-down list.
      ii. When applicable, select the vertex color stream and channel that includes the Inverse Masses data. If data is not provided, then cloth defaults to an inverse mass value of 1.0 for all vertices.
      iii. When applicable, select the vertex color stream and channel that include the Motion Constraints data. If data is not provided, then cloth defaults to a motion constraint value of 1.0 for all vertices.
      iv. When applicable, select the vertex color streams and channels that includes the Backstop Offset and Backstop Radius data. If data is not provided, then no backstop constraints will be applied in the simulation.
d. Choose the **Update** button. **Asset Processor** then updates the asset and includes the cloth data.

5. Configure the cloth component.
   a. Select the cloth mesh node from the drop-down list.
b. Adjust cloth properties to obtain the desired cloth behavior. For more information, see Cloth Component (p. 575).

**View the Cloth Simulation**

In Lumberyard Editor, press **Ctrl+G** or press the **Play** button to run your project.

**Cloth for Actor components**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To use Cloth, you must enable the NVIDIA Cloth gem. For more information, see the NVIDIA Cloth gem (p. 1182) documentation.

You can create cloth assets for entities with Actor components in the content creation application of your choice and import them into Lumberyard from an .fbx file. The actor asset should have the following:
Actor mesh

One or more meshes that visually represent the actor, skinned to a skeleton, that will not be simulated as cloth.

Cloth mesh

One or more meshes that will be simulated and rendered as cloth.
- The cloth mesh must be skinned to bones. The bones do not have to be exclusive to the cloth mesh. The bones must be part of the actor's skeletal hierarchy. Because simulation will drive the cloth mesh, we recommend you use few additional bones for the cloth mesh.
- Cloth data can be added to define per vertex mass and constraint properties using the vertex color tool in your content creation application. For more information see Per vertex properties for cloth (p. 2866).

Skeleton

A skeleton to drive the actor and cloth meshes. Cloth meshes may be skinned to their own bones or any bones in the hierarchy. Bones that drive the cloth meshes must be part of the skeleton's hierarchy.

Animation

A Motion set and Anim graph based on the actor's skeleton. Keyframe animated cloth can be blended with simulated cloth using Motion constraints.

Note

Sample Actor component cloth assets are located in the NVIDIA Cloth gem directory, which is located at /dev/Gems/NvCloth/Assets/Objects/cloth/Chicken/.

For information on exporting actor assets, see FBX Settings actor export (p. 437)

Add Cloth to Actor components

Create cloth by adding the Cloth component to an entity that has an Actor component, and then setting the properties of the Cloth component.

1. In Lumberyard Editor, add a new entity to the level.
2. Add an Actor component to the entity, and reference the actor asset and material.
3. Add an Anim Graph component and reference the actor anim graph asset and motion set.
4. Add a Cloth component to the entity.
5. Set the cloth data of the actor asset.
   a. Click the button beside the Mesh node property to open the FBX Settings window.
b. In the FBX Settings window, on the Actors tab, and choose Add Modifier, Cloth.

c. In the Cloth modifier area:

i. Select the cloth mesh from the drop-down list.

ii. When applicable, select the vertex color stream and channel that includes the Inverse Masses data. If data is not provided, then cloth defaults to an inverse mass value of 1.0 for all vertices.

iii. When applicable, select the vertex color stream and channel that include the Motion Constraints data. If data is not provided, then cloth defaults to a motion constraint value of 1.0 for all vertices.

iv. When applicable, select the vertex color streams and channels that includes the Backstop Offset and Backstop Radius data. If data is not provided, then no backstop constraints will be applied in the simulation.
d. Choose the **Update** button. **Asset Processor** then updates the asset and includes the cloth data.

6. Configure the cloth component.
   a. Select the cloth mesh node from the drop-down list.
b. Adjust cloth properties to obtain the desired cloth behavior. For more information, see Cloth Component (p. 575).

c. You can use the Motion constraints properties Max Distance and Scale to blend between cloth simulation and keyframe animation.

Add cloth colliders to an actor

You can add cloth colliders to an actor to prevent the cloth form penetrating the actor's mesh during simulation. Cloth colliders are added to actors in Animation Editor. For information on adding cloth colliders to an actor, see Add Cloth Colliders to actors (p. 1489).

View the Cloth Simulation

In Lumberyard Editor, press Ctrl+G or press the Play button to run your project.
Per vertex properties for cloth

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Cloth properties can be set per cloth particle using the vertex color tools in your content creation application. In the Cloth modifier, in FBX Settings, you can choose which vertex color stream and which color channel in the stream represents each property. You can use a different stream for each property, or combine multiple properties into a single vertex color stream by storing the properties in different color channels.

Inverse Mass

Inverse Mass calculates a per cloth particle mass value. If no vertex color stream is provided, then the Inverse Mass value of all vertices will be 1.0 by default. The value range for Inverse Mass is 0.0 to 1.0.

An Inverse Mass value of 0.0 excludes the vertex from the cloth simulation. Vertices with a 0.0 Inverse Mass value will be static.
Per cloth particle mass is calculated as \( \text{VertexMass} = \frac{1.0}{\text{InverseMass}} \). For example, if the Inverse Mass value in the color channel is 0.3, then the resulting cloth particle mass value is 3.33. The smaller the Inverse Mass value, the greater the cloth particle mass.

**Motion Constraints**

Motion constraints limit the movement of a simulated cloth particle to an area defined by a sphere. The sphere's position is relative to the corresponding unsimulated vertex position. For a detailed explanation of Motion Constraints, see Cloth simulation constraints (p. 2867).

The Motion Constraints per vertex property calculates the radius of the sphere. Motion Constraints have a value range of 0.0 to 1.0.

A Motion Constraints value of 0.0 constrains the cloth particle to the corresponding unsimulated vertex.

**Backstop**

Backstop prevents a simulated cloth particle from entering an area defined by a sphere. The sphere's position is relative to the corresponding unsimulated vertex position. There are two Backstop properties that you can define per vertex, Backstop Offset and Backstop Radius. For a detailed explanation of Backstop, see Cloth simulation constraints (p. 2867).

**Backstop Offset**

Backstop Offset defines the backstop sphere's offset along the normal of the corresponding unsimulated vertex. The Backstop Offset value range, 0.0 to 1.0, is remapped to a range between -1.0 and 1.0.

A Backstop Offset value of 0.0 is remapped to -1.0 and places the backstop sphere in front of the unsimulated vertex.

A Backstop Offset value of 1.0 places the backstop sphere behind the unsimulated vertex.

A Backstop Offset value of 0.5 is remapped to 0.0 and places the backstop sphere on the unsimulated vertex.

**Backstop Radius**

Backstop Radius calculates the radius of the backstop sphere. Backstop Radius has a value range of 0.0 to 1.0.

A Backstop Radius value of 0.0 disables the backstop for the corresponding vertex.

**Cloth simulation constraints**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Constraints limit the movement of cloth particles to prevent mesh penetration and create more predictable results from cloth simulation. Amazon Lumberyard has two types of cloth constraints: Motion constraints and Backstop.
**Motion constraints** - Constrains a simulated cloth particle within an area defined by a sphere. The sphere's position is relative to the corresponding unsimulated vertex position.

**Backstop** - Prevents a simulated cloth particle from entering an area defined by a sphere. The sphere's position is relative to the corresponding unsimulated vertex position.

Motion constraints and Backstop properties are set by creating vertex color streams for the mesh in a content creation application such as Maya. When exporting a mesh through FBX Settings, you can specify which vertex color streams and color channels define the properties in the Cloth modifier.

Motion constraints and Backstop properties are per vertex and can be used with Actor and Mesh components.

### Motion constraints

Motion constraints limit the movement of a simulated cloth particle to an area defined by a sphere. The sphere is centered on the corresponding vertex of the unsimulated mesh. The radius of the sphere is calculated using the Motion Constraints value from the vertex color stream and the Max Distance, Scale and Bias properties in the Motion constraints property group of the Cloth component.

If no vertex color stream is present, then a default value of 1.0 is used for the Motion constraints of each vertex. Motion Constraints vertex color stream values range from 0.0 to 1.0. The simulated cloth particles are fully constrained to the unsimulated mesh vertices if the vertex color stream has a value of 0.0.

The diagram below visualizes Motion constraints.

### Backstop

Because cloth colliders are simple primitives, you may encounter scenarios where cloth colliders aren't sufficient to prevent the simulated cloth particles from penetrating other meshes. You can use Backstop to fine tune the behavior of simulated cloth particles to address these scenarios.

Backstop prevents a simulated cloth particle from entering an area defined by a sphere. The sphere is positioned along the normal of the corresponding unsimulated mesh vertex through an offset property.
Backstop requires color channels from a vertex color stream to define both the radius of the sphere and the sphere's offset. The Backstop Radius and Backstop Offset values in the vertex color stream are scaled by the Radius and Offset values in the Cloth component Backstop properties.

Because a vertex color channel can only contain values between 0.0 and 1.0, the value in the color channel representing the Backstop Offset property will be remapped to a range between -1.0 and 1.0.

A Backstop Offset vertex color value of 1.0 is scaled by the Back Offset property of the Cloth component. The backstop sphere is placed behind the unsimulated mesh vertex.

A Backstop Offset vertex color value of 0.0 is remapped to -1.0 and scaled by the Front Offset property of the Cloth component. The backstop sphere is placed in front of the unsimulated mesh vertex.

A Backstop Offset vertex color value of 0.5 is remapped to 0.0 and scaled by the Back Offset property of the Cloth component. The backstop sphere is placed on the corresponding unsimulated vertex.

The Backstop Radius vertex color channel has values in the range of 0.0 to 1.0. A value of 0.0 in the Backstop Radius vertex color channel disables the backstop for the corresponding vertex.

The diagram below visualizes Backstop.

\[
\begin{align*}
\text{Cloth vertex position } C_p & \text{ will be constrained during simulation to remain outside the sphere with position } S_p \text{ and radius } S_r \\
S_r & = Vbr \ast Radius \\
S_p & = \begin{cases} 
Vp - Vn \ast (S_r + Vbo \ast \text{BackOffset}) & \text{when } Vbo \geq 0 \\
Vp + Vn \ast (S_r - Vbo \ast \text{FrontOffset}) & \text{when } Vbo < 0 
\end{cases}
\end{align*}
\]

**Cloth Component - Backstop properties**

- Backstop
  - Radius: 0.3 m
  - Back offset: 0.0 m
  - Front offset: 0.0 m

**Cloth visual debugger**

This feature is in preview release and is subject to change.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To enable debug cloth visualization, use the following console variables (CVARs) in the editor console.

**cloth_DebugDraw <value>**
- Draw the cloth mesh wireframe.
  - 0: Disable wireframe display.
  - 1: Enable wireframe display.

**cloth_DebugDrawNormals <value>**
- Draw the cloth mesh normals.
  - 0: Disable normals display.
  - 1: Enable normals display.
  - 2: Enable normals, tangents and bitangents display.

**cloth_DebugDrawColliders <value>**
- Draw the cloth colliders.
  - 0: Disable collider display.
  - 1: Enable collider display.

**cloth_DebugDrawMotionConstraints <value>**
- Draw the cloth motion constraints.
0: Disable motion constraint display.

1: Enable motion constraint display.

cloth_DebugDrawBackstop <value>

Draw the cloth backstop.

0: Disable backstop display.

1: Enable backstop display.

GPU cloth simulation with NVIDIA CUDA

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

NVIDIA Cloth gem supports NVIDIA CUDA on Windows 10. With NVIDIA CUDA enabled for cloth, the simulation is computed on the GPU instead of the CPU. This allows you to use all the computational power of the GPU to run more complex cloth instances, in larger numbers, leaving the CPU free to do other tasks.

For more information about NVIDIA CUDA, visit the NVIDIA CUDA portal.

Contents

- GPU cloth simulation requirements (p. 2871)
- Installing NVIDIA CUDA Toolkit with Setup Assistant (p. 2871)
- Enable NVIDIA CUDA for cloth simulation (p. 2872)
- Releasing your project with NVIDIA CUDA (p. 2873)

GPU cloth simulation requirements

- Microsoft Windows 10
- NVIDIA CUDA Toolkit 10.1.168 or newer
- NVIDIA graphics card with CUDA support - Visit the NVIDIA CUDA GPU support portal for information.
- NVIDIA driver version 418.96 or newer - The latest driver version is recommended.

Installing NVIDIA CUDA Toolkit with Setup Assistant

NVIDIA CUDA Toolkit can be installed through Setup Assistant. The install link is located on the Install Software page, in the Optional software group.

1. Run Setup Assistant.
2. Choose Install Software and scroll down to the Optional software group.
3. Choose Get it to the right of NVIDIA CUDA Toolkit.
4. A browser opens on the NVIDIA CUDA Toolkit Download page. Use the prompts on the page to download the correct version of the toolkit for your system.

5. Run the NVIDIA CUDA Toolkit installer and follow its prompts for Express installation.

   **Note**
   If NVIDIA CUDA Express installation fails, then try these remedies:
   - Run the installation again and choose a **Custom Installation** this time, only selecting the following elements: CUDA Development Compiler and CUDA Runtime Libraries.
   - Update the graphics card driver to the latest version and retry installation.
   - If you are using a laptop, then the manufacturer's drivers for the graphics card might not be compatible with CUDA. Check the manufacturer's website for the latest drivers.

6. Click the **refresh** button at the top-right corner of Lumberyard Setup Assistant. If the NVIDIA CUDA Toolkit is detected, then it appears as **Installed**.

---

**Enable NVIDIA CUDA for cloth simulation**

To use NVIDIA CUDA for cloth simulation on the GPU, you must configure and rebuild your project, and enable NVIDIA CUDA in your PC configuration file.

1. Use Project Configurator to add the **NVIDIA Cloth Gem** to your project.

2. Configure your project. Use the following command.

   **Note**
   Open a new command prompt and navigate to `lumberyard_version\dev\` to run the commands below. Do not use an existing command prompt.

   ```
   lmbr_waf configure
   ```

3. Check in the output that it has detected NVIDIA CUDA.

   ```
   [INFO] Detected NVIDIA CUDA SDK at: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1
   ```
4. Build your project. Use the following command.

```bash
lmbr_waf build_win_x64_vs2019_profile -p all --progress
```

5. Add the following line to the system_windows_pc.cfg file located in `lumberyard_version\dev`.

```
cloth_EnableCUDA = 1
```

6. Run Lumberyard Editor. This also runs the Asset Processor in the background.

7. Verify that the NVIDIA Cloth Gem is running cloth simulation on the GPU by looking for the following line in the game log or in the editor console output:

```
NVIDIA NvCloth Gem using GPU (CUDA) for cloth simulation
```

**Releasing your project with NVIDIA CUDA**

End users do not require the NVIDIA CUDA Toolkit to be installed on their PC in order to run your release builds.

Cloth simulations automatically run on the GPU, if a GPU that is compatible with CUDA is present. If an NVIDIA CUDA compatible GPU is not available, then the simulation defaults to running on the CPU.

For more information about release builds, see Building Lumberyard projects (p. 61).
Creating and Customizing Project User Interfaces

You can use the **UI Editor** to create and customize various parts of the game user interface, such as images, text, buttons, menus, scroll boxes, and heads-up displays (HUDs). For a tutorial about UI creation for games, see Lumberyard Tutorials.

**Topics**
- Using the UI Editor (p. 2874)
- Working with UI Canvases (p. 2888)
- Defining Game and Level Load Screens (p. 2910)
- Working with UI Slices (p. 2912)
- UI Elements (p. 2916)
- UI Components (p. 2923)
- Implementing New Fonts (p. 2999)
- TrueType Fonts (p. 3009)
- Animating the UI (p. 3011)
- UI Lua Reference (p. 3032)

**Using the UI Editor**

You can use the **UI Editor** to create, customize, and animate various game user interface elements and components such as menus, buttons, and heads-up displays (HUDs).

The **UI Editor** consists of the following:

1. **Toolbar** – Commonly used tools and settings
2. **Hierarchy pane** – List of UI elements that you create
3. **UI canvas tab bar** – Tabbed display of open canvases
4. **Viewport** – Display of the UI elements on the current UI canvas
5. **Properties pane** – Component properties for the selected element
6. **Animation Editor** – Tool for animating UI elements

**Note**
You can tear away and redock the **Hierarchy pane**, the **Properties pane**, the **Animation Editor**, and sections of the toolbar to customize the **UI Editor**.
To open the UI Editor

- In Lumberyard Editor, choose **Tools** and then **UI Editor**.

Topics

- Toolbar (p. 2875)
- Rulers and Guides (p. 2878)
- Asset Drag and Drop (p. 2881)
- Sprite Editor (p. 2882)

## Toolbar

### Interaction Modes

The **UI Editor** toolbar features interaction modes. The selected interaction mode determines how you can interact with the UI elements in your canvas.

#### Interaction modes

**Select**

Select elements in your canvas. In this mode, selecting is the only action that you can use on your UI elements.
Move
Select and move UI elements. Moving a UI element modifies its offset. In the Move mode, you can view but not interact with the anchor widget, which is disabled.

You can move UI elements in the following ways:
- Select an element and drag it to a new position.
- Select a particular axis (X, Y, or Z) on the axis gizmo and drag the element by only one axis at a time.
- Nudge your selected element using keyboard arrows.
- Select multiple elements and align them using the Alignment Tool (p. 2876).

Rotate
Select and rotate UI elements.

Resize
Select and resize UI elements. Resizing modifies the UI element’s offsets.

Anchor
Select and move UI elements by their anchors. If you select one element, the anchor widget appears in blue, and you can interact directly with that element. If you select multiple elements, the anchor widget is disabled.

In Anchor mode, you can interact with UI elements in the same ways as in the Move mode.

Alignment Tool
The alignment tool is a set of buttons on the UI Editor’s toolbar.

The alignment tools are enabled when your interaction mode is Move or Anchor and you have selected two or more movable elements.

Note
One type of element that you can’t moved is one that a parent layout element controls.

With these tools, you can align elements by lining up their:
- Top edges
- Elements’ horizontal centers
- Bottom edges
- Left edges
- Elements’ vertical centers
- Right edges

To align objects
1. Select one of the following interaction modes:
   - Move
• **Anchor**

2. On your canvas, select two or more movable elements.

   The alignment tools on the toolbar are now enabled.

3. Select an alignment button.

The method that the alignment tool uses to move the elements depends on the interaction mode you're using. If the mode is in **Move**, elements are moved using their offsets. If the mode is **Anchor**, the elements are moved by their anchors.

A bounding rectangle is a useful graphic for understanding how the alignment tool determines how it aligns elements.

Imagine a gray bounding rectangle that encompasses the elements in it.

![Diagram](image)

If you align vertically by the elements' centers, the elements are centered at the bounding rectangle's center, which remains in its original position.
If you align vertically by the elements' right edges, the elements' right edges are aligned to the bounding rectangle's right edge.

**Rulers and Guides**

Use **Rulers** and **Guides** to visually guide the placement of your UI elements on your UI canvas.

**To show or hide Rulers**

- Do one of the following:
  - Press **Ctrl+R**
  - Choose **View** and then **Rulers**
The ruler's units are measured in pixels in the canvas space. Magenta lines on the rulers mark the current location of your cursor.

Guides appear as green lines and act as a visual aid for positioning UI elements.

**To show or hide Guides**

- Do one of the following:
  - Press Ctrl+; (semicolon)
  - Choose View and then Guides

You place the guides at a specific pixel offset on a canvas. The UI Editor displays guides as green lines, which you can position UI elements around or along.

**To create a guide**

1. Ensure that the Rulers appear.
2. Click on the top or side ruler and drag into the canvas.
   
   If you start in the top ruler, you create a horizontal line. Starting in the side ruler creates a vertical line.
3. Release at the position that you want to place the guide.
4. To adjust the position of the guide, click the guide and drag it to a new position.

![Guide adjustment](image)

**Note**
You must be in **Move** or **Anchor** mode to adjust a guide's position.

You can lock guides to protect against unintentionally moving them. Locking the guides also makes it easier to move UI elements across and around the guides.

**To lock guides**

- Choose **View** and then **Lock Guides**.

**To delete one guide**

- Click on the guide and drag it off of the canvas.
To delete all guides

- Choose View and then Clear Guides.

You can change the color of all guides.

To change guide color

1. Click in an empty area on the viewport or hierarchy pane so that no UI elements are selected, and you see the UI Canvas component in the Properties pane.
2. In the UI Canvas component, under Editor settings, click the Guide color and select a new color.

Asset Drag and Drop

You can drag and drop existing assets from the Asset Browser (p. 296) into the UI Editor’s viewport, hierarchy pane, properties pane, or a specific property field. This can be a convenient way to create new entities and assets to work with in the UI Editor.

Dragging assets associated with components

- You can drop any asset that is associated with a component, such as a Script Canvas file or a Lua script file, into the UI Editor viewport or hierarchy pane.

When you drag an asset from the Asset Browser into the viewport, the UI Editor does the following:

- Creates a new entity at the cursor's location.
- Adds the associated component, which is indicated by the icon next to the asset in the Asset Browser.
- Assigns the asset for that component's property.

When you drag an asset from the Asset Browser into the hierarchy pane, the UI Editor does the following:

- Creates a new entity and places it as a child of an existing parent, unless the drop location is directly on an entity. In this case, the associated component is added to that existing entity without creating a new one. To override this behavior, press the Shift key down while you drop the asset. This creates a new entity and places it as a child of the existing entity.
- Adds the associated component, which is indicated by the icon next to the asset in the Asset Browser.
- Assigns the asset for that component's property.

When you drag an asset from the Asset Browser into the properties pane, the UI Editor does the following to all selected entities:

- Adds the associated component, which is indicated by the icon next to the asset in the Asset Browser.
- Assigns the asset for that component's property.

Dragging UI slice assets

- You can drop any slice asset that consists of only UI entities into the UI Editor viewport or hierarchy pane.
When you drag a UI slice asset from the Asset Browser into the viewport, the UI Editor does the following:

- Instantiates a new entity from the dragged slice at the cursor's location.

When you drag a UI slice asset from the Asset Browser into the hierarchy pane, the UI Editor does the following:

- Instantiates a new entity from the dragged slice and places it as a child of an existing parent, according to the drop location in the hierarchy.

### Dragging UI Canvas assets

- Drag UI Canvas assets from the Asset Browser into the UI Editor to open them for editing.

### Dragging assets to property fields

- Drag an asset from the Asset Browser onto an asset property field in the properties pane to assign that asset to a component's property.

## Sprite Editor

The Sprite Editor configures the following sprite configurations:

- Border values for sliced image types (p. 2883)
- Sprite sheets (p. 2886)

You open the Sprite Editor from the Image component's properties.

### To open the Sprite Editor

1. Open the UI Editor as explained in the preceding section.
2. Choose the ellipsis button next Sprite path and select the sprite file.
3. To the right of Sprite path, click the arrow icon.

The Sprite Editor has the following features:
• **Sprite viewport** – Displays sprite image.

• **Border manipulators** – Sets border properties for sliced image types. To adjust the borders, drag the dotted lines, which are called manipulator positions. Changing these positions updates the corresponding **Border Properties** values.

• **Properties**
  
  • **Image resolution** – The size of the image.
  
  • **Alias** – A short description of what the cell represents. Use this setting to improve the readability of sprite-sheet index values. You can use the same alias string for multiple sprite sheet cells.
  
  • **Top, Bottom, Left, Right** – The number of pixels from the respective edge of the image where the sliced region is situated.

• **Configure Spritesheet** – Available only for sprites that are not currently configured as a sprite sheet. For more information, see Configuring Sprite Sheets (p. 2886).

### Sliced Image Type

Lumberyard uses slice resizing to resize certain images intelligently. For example, using regular resizing to widen the following image results in distortion of the corners and edges.
Slice resizing divides an image into nine sections that scale in such a way as to preserve border and corner details. This technique avoids the distortion that occurs with typical image scaling.

Each area is resized as follows:

- The center is resized both horizontally and vertically
- The corners aren't resized at all
- The top and bottom edges are resized horizontally only
- The right and left edges are resized vertically only

Slice resizing is useful for images with borders and corner details, such as buttons with rounded corners.
Using the Sprite Editor (p. 2882), you can manipulate where the sections are by dragging the dotted lines on the preview image.

**Tip**
You can see your changes in real time. To do this, before you open the Sprite Editor, change the Image component's ImageType property to Sliced.
Sprite Sheets

You can configure an image as a sprite sheet.

A sprite sheet is a collection of separate images—such as icons, buttons, and other UI assets—that are stored in a single image. Although you can keep all of your images in separate files, using a sprite sheet has several advantages:

- Faster performance – For that collection of assets, Lumberyard can load just one image from disk instead of many separate images. Loading multiple images requires many hard drive seeks and is performance expensive.
- Workflow improvements – When your animation contains multiple frames, for example, it's easier to manage just one image that contains all of the animation frames rather than separate files. This makes editing and other workflows easier.

Other workflow improvements include easier management of your assets. For example, you could have one image called `mainmenu_ui_assets_spritesheet.png` that contains all of your buttons rather than a series of files such as `mainmenu_button1.png`, `mainmenu_button2.png`, and so on.

Example

The following image contains 12 walking images in a single row.

In the following procedure, you can divide the image into 12 columns when configuring the sprite sheet. You can then select the piece of the image that you want to display for your Image component.
To configure an image as a sprite sheet

1. In the Sprite Editor (p. 2882), click Configure Spritesheet in the lower-left corner.

   The Configure Spritesheet view displays two new sections, Configure Spritesheet and Select cell.

   ![](image.png)

   - Configure Spritesheet
     - Rows: 1
     - Columns: 12

   - Select cell
     - Multiple walking images are displayed, with one highlighted.

   - Border Properties
     - Cell size is 218 x 218
     - Alias: 
     - Top: 0.0 px
     - Bottom: 0.0 px
     - Left: 0.0 px
     - Right: 0.0 px

   - Save, Cancel

2. Enter the number of rows and columns. The walking images example has 12 columns and 1 row. These values divide the sprite into a uniform grid and assume that each cell of the sprite sheet is of the same (uniform) size.

3. In the Select cell section, click a cell to select it and display its properties.

4. To configure individual cells with slices scaling, drag the dotted lines to the preferred positions. The Top, Bottom, Left, and Right properties are updated automatically to reflect the current positions.

   For more information about slice scaling, see Sliced Image Type (p. 2883).
5. Click **Save** to save your changes and close the **Sprite Editor**. Or click **Cancel** to revert your changes and close the **Sprite Editor**.

6. To select the specific cell of the sprite sheet that you want to use, in the **Image** component properties, select the appropriate **Index** number.

   The **Sprite Editor** assigns index numbers in the rows and columns of the sprite sheet, from left to right and then top to bottom, starting with 0 (zero).

   If you defined an **Alias** in the **Sprite Editor** properties, that also appears next to the index numbers.

![Image properties](image.png)

   The cell that you selected appears in the **UI Editor** viewport.

---

**Working with UI Canvases**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/blogs/gaming/3d-engine/) or visit the [AWS Game Tech blog](https://aws.amazon.com/blogs/gaming/) to learn more.

The **UI Editor** uses the concept of a canvas as an invisible backdrop for your game user interface elements. Once you create a canvas, you can add elements such as images, text, and buttons.

**To create a UI canvas**

1. In Lumberyard Editor, click **Tools, UI Editor**.

   The **UI Editor** automatically creates an empty canvas.

2. In the **UI Editor**, add elements (p. 2916), components (p. 2923), slices (p. 2912), and prefabs (p. 2920).

3. Click **File, Save As**. Name the canvas with a `.uicanvas` file extension, and then click **Save**.

**Topics**

- [Navigating the Viewport](p. 2889)
- [Changing the Canvas Size](p. 2890)
- [Previewing Canvases](p. 2891)
- [Configuring Canvas Properties](p. 2894)
- [Loading Canvases in Lua](p. 2897)
- [Placing UI Canvases in the 3D World](p. 2897)
• Using Texture Atlases (p. 2900)
• Debugging UI Canvases (p. 2906)

You can open multiple canvases in the UI Editor. For each open canvas, a tab appears in the tab bar. The active canvas is highlighted in the tab bar. Its content is displayed in the viewport and its elements and properties are displayed in the hierarchy and properties panes.

From the UI Editor's File menu, in addition to the basic open, close, and save functionality, you can also do the following:

• Save all open canvases
• Close all open canvases
• Close all but the active canvas

Navigating the Viewport

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. **Download O3DE** or visit the **AWS Game Tech blog** to learn more.

The UI Editor features a rectangle with a checkerboard pattern on a dark gray background.
The checkerboard pattern represents empty space within the UI canvas, and the dark gray represents the space outside of the canvas. Anything within the UI canvas space is visible when the canvas is loaded.

**To zoom in or out on a UI canvas**

Do one of the following:

- Mouse – Scroll the mouse wheel.
- Keyboard – Press Ctrl + or Ctrl -.
- Menu – Click View, then click Zoom In or Zoom Out.
- Toolbar – In the Zoom box, enter a percentage value or click the up and down arrows.

**To pan the view on a UI canvas while dragging the canvas with the left mouse button**

- With the mouse on the UI canvas, drag using the middle mouse button.
- Press and hold the space bar while dragging the canvas.

**To toggle common zoom settings**

Do one of the following:

- Fit canvas to current view (default) – Press Ctrl+0, or click View, Fit Canvas.
- View canvas at actual size – Press Ctrl+1, or click View, Actual Size.

---

**Changing the Canvas Size**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Previewing Canvases

Change your game UI canvas size to visualize how your canvas might look on other displays and devices of varying resolutions. The size at which you save your canvas is the reference size that is used when you perform the Scale to Device action.

**To change the canvas size**

1. On the toolbar, click the arrow beside the resolution to see a list of commonly used canvas sizes for various platforms.

2. Select the size you want or click **Other** to enter a custom canvas size.

---

**Tip**

You can customize the list of canvas sizes that appear in the list by modifying a JSON file stored locally on your machine. In Windows, the canvas size presets file is located in the following directory:

`C:\Users\<UserName>\AppData\Local\Amazon\Lumberyard\size_presets.json`

---

**Previewing Canvases**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

You can preview your game UI canvas to visualize how it might look at different screen resolutions and to see how the interactive elements change states.

**Topics**
• Setting Canvas Size in Preview (p. 2893)
• Previewing Canvas Behavior (p. 2893)

The UI Editor Preview consists of the following:

1. **Toolbar** – Tools to view the current **Viewport size**, **Preview canvas size** (selectable), and **Canvas scale**.
2. **Viewport** – Display of the UI canvas as it would appear at the selected resolution.
3. **Animation List (p. 2893)** – List of the animation sequences in the canvas, which you can control using the playback toolbar. Close this pane to increase the viewport size. Use the **View** menu to restore it.
4. **Action Log (p. 2893)** – Record of actions triggered by the canvas's interactable elements. Close this pane to increase the viewport size. Use the **View** menu to restore it.

To open UI canvas Preview

Do one of the following:

• From the **UI Editor** toolbar, click **Preview**.
• From the **UI Editor** menu, click **Preview, Preview**.
• Press **Ctrl+P**.

To exit the canvas preview, click **End Preview**.
Setting Canvas Size in Preview

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Change your game UI canvas size in **UI Editor Preview** to visualize how your canvas might look at different screen resolutions and to see how the interactive elements change state. Changing your canvas size in **Preview** does not affect the canvas size at which you are authoring the UI canvas—that is controlled in the **UI Editor** (p. 2890).

Setting your canvas size in **Preview** is useful when designing games that run on devices that have multiple resolutions. You can see at different resolutions how an element's size and position changes based on the settings of its **Transform2D** properties, such as anchors, offsets, and the **Scale to Device** settings. For more information on the **Transform2D** properties, see **Transform2D Component** (p. 2926).

The **Canvas scale** in the toolbar shows the scale at which the canvas is displayed. If the **Preview canvas size** selected is larger than the viewport size, the canvas you are previewing is drawn at a reduced scale.

Previewing Canvas Behavior

In **UI Editor Preview**, the UI elements in your canvas perform as they would when the game is running.

Try these examples:

- Pause on an interactive element to show its hover state.
- Press (click) an interactive element to show its pressed state.
- Adjust sliders.
- Input and edit text.
- Use keyboard, mouse, or gamepad to interact with the UI.

**Note**

If the interactive component's **Input enabled** setting is deselected (unchecked), that element is drawn in its disabled state and does not respond to hover or click actions.

Animation List

The **Animation List** pane lists all the UI animation sequences found on the canvas that you are previewing. Select an animation to use the reset, play, pause, and set-to-end controls. Hold **Ctrl** or **Shift** to select and control multiple animations at once. You can also control animations independently and simultaneously so that one may be playing, for example, while you pause another.

Action Log

The **Action Log** pane shows the actions generated by interacting with interactive elements in the UI canvas while in **Preview**. These logged actions help the canvas designer ensure that correct actions are being triggered.
To use this feature, you must type text strings in the **Actions** section of the interactive element's properties.

**To enable Action Log entries**

1. In the **UI Editor** viewport or **Hierarchy** pane, select the element to which the interactive component is attached.
2. In the **Properties** pane, under the **Actions** category, enter a text string for each action for which you want to trigger an action log entry.

   The text strings are fully customizable; you can enter any string that helps you ensure that the correct actions are being triggered.

For example, in the picture below, **EnablerChanged** is displayed whenever the **Enable Input** check box changes state (from off to on, or on to off). **EnablerOn** is displayed when the check box is selected, and **EnablerOff** is displayed when it is deselected.

![Example of Action Log entries](image)

During **Preview**, Script Canvas and Lua scripts aren't active. Actions taken in UI canvas **Preview** have no effect on anything outside of the canvas.

**Configuring Canvas Properties**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/developer-preview/) or visit the [AWS Game Tech blog](https://aws.amazon.com/game-tech/) to learn more.

The canvas properties are displayed in the **UI Editor Properties** pane when no elements are selected.
Rendering Properties

The following properties define how a canvas is rendered:

- **Draw order** – The value of this property determines the order that this canvas draws relative to other canvases. Higher numbers draw on top of lower numbers. When canvases have the same draw order, Lumberyard draws them in the order that they are loaded.

- **Is pixel aligned** – Selected by default. This property makes textures look sharper by rounding the position of the elements' corners to the nearest exact pixel. For example, if the position of a corner of an element rectangle is at 123.45, 678.90, then it is rounded to 123.00, 679.00.

- **Is text pixel aligned** – Selected by default. This property makes text look crisper by rounding text positions down to the nearest pixel. An exception to this rule occurs when fonts have been scaled down, in which case the text position is rounded to the nearest pixel. If the property is not checked, the text position is not rounded. You might consider unchecking this property if, for example, a text element will animate or move.

- **Render to texture** – Cleared by default. When this property is selected, the canvas is drawn to a texture rather than to the screen. When you select this property, you are prompted to enter a **Render target name** for the texture. You can enter any name, but the convention is to prefix the name with the `$` symbol to distinguish it from texture assets.

Input Properties

The following properties define how a canvas handles inputs:

- **Handle positional** – Selected by default. This property causes an automatic response to positional input such as mouse movement, mouse button clicks, and touch screen input. Keyboard inputs also cause an automatic response when an interactive UI element is active (such as an elemental with a **Text Input** component on it).
Configuring Canvas Properties

Common reasons to de-select this property are if the canvas doesn't require input, or if you configure your game to handle all inputs and pass selected inputs to the UI system.

- **Consume all input** – Cleared by default. When this property is selected, this canvas consumes *all* input events while it's enabled, regardless of whether the canvas handles a specific input event. For example, if you have canvas A covering canvas B, you probably don't want canvas B handling any input while canvas A is obstructing it, so you would select this property on canvas A. Modal dialog boxes are another example of a canvas that should have this property selected.

Note that anytime a canvas is loaded, if it's set to consume all input, then it steals the inputs from any other loaded canvas. This includes canvases that are set to consume all inputs themselves.

- **Handle multi-touch** – Cleared by default. When this property is selected, it enables elements on the canvas to handle multi-touch input. This is useful for handling input from touch-based screens, such as mobile devices.

- **Handle navigation** – Cleared by default. When this property is selected, it causes an automatic response to navigation input. For example, on a PC, pressing arrow keys moves focus from one interactive UI element to the next, and pressing Enter activates an interactive UI element. We recommend de-selecting this property for canvases that are placed in the game world.

- **Navigation threshold** – The analog input value, from a thumbstick, for example, that must be exceeded before a navigation command is processed. Valid ranges are decimal values between 0 and 1, and the default is 0.4. Adjust this value based on the input sensitivity needs of your UI.

- **Navigation repeat delay** – The delay, in milliseconds, before a held navigation command begins repeating. The default is 300 ms. Adjust this value based on the needs of your UI.

- **Navigation repeat period** – The delay, in milliseconds, after the initial repeat delay, before a held navigation command repeats again. The default is 150 ms. Adjust this value based on the needs of your UI.

For example, if you had a menu list where you hold a button to navigate to the next item in the list, the navigation property settings are used as follows:

1. Hold down the button past the *navigation threshold* to navigate to the next item.
2. Continue holding for an amount of time equal to the *navigation repeat delay* to navigate a second time.
3. Continue holding for an amount of time equal to the *navigation repeat period* to navigate a third time. Thereafter, as you continue holding the button, you will navigate again, every time an amount of time equal to the navigation period elapses.

- **First focus element** – Displayed when **Handle navigation** is selected. First focus element specifies which element gains focus when a canvas is first loaded and a mouse is not detected. For more information about element navigation, see First Focus Element (p. 2952).

### Tooltips Properties

The following property defines how a canvas displays tooltips:

- **Tooltip display element** – Controls which element the game displays when your users hover over an interactive element. Select an element from the drop-down list. This list is composed of the elements on your current canvas that contain the TooltipDisplay component. For more information about the Tooltips components, see Tooltip Components (p. 2985).

### Editor Settings Properties

The following properties define UI Editor behavior:
**Snap distance** – The distance between positions on the grid when **Snap to grid** is selected in the toolbar.

**Snap rotation** – The number of degrees between each step of rotation when you use the rotation gizmo to rotate an element in the viewport when **Snap to grid** is selected in the toolbar.

**Guide color** – The color of the guide lines on this canvas. For more information about using guides in UI Editor, see Rulers and Guides (p. 2878).

**Texture atlases** – The texture atlas that this canvas loads. Using a texture atlas can reduce the number of draw calls in certain situations, resulting in better performance for your UI. For more information about texture atlases, see Using Texture Atlases (p. 2900).

## Loading Canvases in Lua

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the Lua scripting language to load and unload UI canvases. For more information, see Loading Canvases in Lua in the Amazon Lumberyard Legacy Reference.

## Placing UI Canvases in the 3D World

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can place a UI canvas directly on an object in the 3D world, as opposed to showing it in screen space. To do this, you render a UI canvas to a texture, and then use that texture in a material on a 3D mesh.

You can use any material on any type of entity to display a texture rendered by a UI canvas. However, if players are to interact with the UI canvas in the 3D world—by clicking with the mouse, for example—you must use a component entity.

To see an example of a UI canvas on an object in a 3D world, open the *UiIn3DWorld* level in the Samples Project.

Follow all the steps in the following procedure if you need to create a canvas that players can interact with. If the canvas is not to be interactive, then you only need steps 1 through 5.

### To place a UI canvas on an object in the 3D world

1. Create your UI canvas file (p. 2888). In the canvas properties (p. 2894), select **Render to texture** and enter a name in the **Render target** text box. You can enter any name, but the convention is to prefix the name with the * character to distinguish it from other texture assets.

2. In the level, create a component entity (p. 463).

3. In the **Entity Inspector**, add to this component entity (p. 480) a **UI Canvas Asset Ref** to specify the UI canvas and optionally to load it automatically when the level loads.

4. In the **Material Editor** (p. 1721), create a material that uses the render target texture that is rendered by your canvas. Under **Texture Maps**, for **Diffuse**, specify the texture file name.

5. Add a **Mesh** (p. 684) component to the component entity and choose the mesh asset onto which you want to map your canvas. Use the **Material override property** to select the material that you created.
6. Add a **Mesh Collider** and a **Static Physics** component. Physics is required on this entity because a ray cast is used to translate a mouse or touch input into a position on the UI canvas that is at that point in the world.

7. Add a **UI Canvas on Mesh** component. Type a canvas name in the **Render target override** property if you want to load several instances of the UI canvas on different meshes and have them display different states. Otherwise, leave this property blank.
### Placing UI Canvases in the 3D World

In the Entity Inspector, you can see the settings for placing a UI Canvas in the 3D world.

- **Name**: `UICanvas_WidgetTestNotLit`
- **Status**: `Start active`

#### Transform
- **Parent entity**: 
- **Translate**: 
  - X: 66.0 m
  - Y: 73.0 m
  - Z: 32.5 m
- **Rotate**: 
  - X: 40.0 deg
  - Y: 0.0 deg
  - Z: 0.0 deg
- **Scale**: 
  - X: 1.0
  - Y: 1.0
  - Z: 1.0
- **Parent activation**: `Original relative transform`
- **Static**: 
- **Network Sync**: 
  - Sync to replicas: 
  - Position Interpolation: `None`
  - Rotation Interpolation: `None`

#### UI Canvas Asset Ref
- **Canvas pathname**: `WidgetTest`
- **Load automatically**: 

#### Mesh
- **Visible**: 
- **Mesh asset**: `plane_proxy_001`
- **Material override**: `uiCanvasMaterial_WidgetTestNotLit`

#### Mesh Collider

#### Static Physics
- **Enabled initially**: 

#### UI Canvas on Mesh
- **Render target override**: `$UIWidgetTestNotLit`

#### Lua Script - LoadCanvasIntoEntity
- **Bind To network**: 
- **Properties**: 
- **Script**: `LoadCanvasIntoEntity`
Using Texture Atlases

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To reduce draw calls, you can add individual textures to a texture atlas and then add one or more texture atlases to a UI canvas.

Characteristics

UI canvases and texture atlases have the following characteristics:

- Each UI canvas contains a list of texture atlases to be loaded.
- Texture atlases are loaded when the UI canvas loads. They're unloaded when the UI canvas unloads.
- If multiple UI canvases load the same texture atlas, the texture atlas is loaded only once.
- UI elements that render textures preferentially use textures from a loaded texture atlas, if available.
- The texture atlas is only unloaded when all UI canvases that loaded it are unloaded.

Advantages

The principle advantages of texture atlases are the following:

- Draw calls are significantly reduced.
- Enhanced compression. Although source textures whose dimensions aren't multiples of four aren't compressed, texture atlases are always compressed.
- Forced preloading of texture groups. When the canvas loads, it also loads its texture atlases.

To see examples of texture atlases, open the UI_TextureAtlasSample level in the Lumberyard SamplesProject.

Topics

- Creating a Texture Atlas (p. 2900)
- Adding Texture Atlases to a UI Canvas (p. 2902)
- Using Texture Atlases to Reduce UI Draw Calls (p. 2904)

Creating a Texture Atlas

To create a texture atlas, you create a texture atlas (.texatlas) file. A texture atlas file is a text file that specifies the image files to be added to the texture atlas. The Asset Processor automatically builds files that have the .texatlas extension.

Texture Atlas File Format

Each line of the .texatlas file represents a command that is executed sequentially. Empty lines are ignored.

Comments

Any line that starts with // denotes a comment.
Example

```
// This is a comment that is ignored.
```

Properties

A line that has an equals (=) sign is considered a property value assignment. Properties not specified use their default values.

The following table shows the list of configurable properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxdimension</td>
<td>4096</td>
<td>The maximum width and height of the output texture atlas.</td>
</tr>
<tr>
<td>padding</td>
<td>1</td>
<td>The minimum number of extra pixels around each texture in the texture atlas. For compression purposes, the edge pixels of each texture are duplicated. The amount of duplication is determined by the calculation <code>image_size + padding</code> rounded up to the nearest compression unit of four.</td>
</tr>
<tr>
<td>poweroftwo</td>
<td>false</td>
<td>Whether the width and height of the output texture atlas is a power of two. If PVRTC compression is used for iOS, the output texture is a power of two regardless of this setting.</td>
</tr>
<tr>
<td>square</td>
<td>false</td>
<td>Whether the width and height of the output texture atlas is the same. If PVRTC compression is used for iOS, the output texture is a square regardless of this setting.</td>
</tr>
<tr>
<td>unusedcolor</td>
<td>#3CB371FF</td>
<td>The color for the unused space in the output texture atlas.</td>
</tr>
<tr>
<td>whitetexture</td>
<td>true</td>
<td>Whether to include a white texture with a path name of WhiteTexture in the output texture atlas.</td>
</tr>
<tr>
<td>presetname</td>
<td>TextureAtlas</td>
<td>The preset to use for image processing. If TextureAtlas is used as the presetname, either explicitly or by default, images will be compressed. This compression can lead to loss of image quality.</td>
</tr>
</tbody>
</table>

When you assign property values, note the following:

- Whitespace is allowed.
- Properties and values aren’t case sensitive.
- If a property value is assigned twice, only the last assignment is accepted.
- The following entries report an error to the Asset Processor and fail the asset processing job:
  - Unrecognized properties
  - Properties with incorrect values
  - Lines with more than one equal (=) symbol

File Paths
If a line specifies a path to an image file, the image is included in the texture atlas. Image file paths can be relative to any watch folder that the Asset Processor monitors for assets. If a line refers to a file that can't be loaded, an error is reported to the Asset Processor and the asset processing job fails. Lines that are neither comments nor properties are assumed to be image file paths.

**Example**

```
UI/Textures/LyShineExamples/button.tif
```

**Updating a Texture Atlas**

The Asset Processor automatically rebuilds the texture atlas if a texture in the atlas changes or if an existing source `.texatlas` file changes.

**Texture Atlas Output Files**

The Asset Processor outputs two files that represent a texture atlas: a `.dds` file and a `.texatlasidx` file. The `.dds` file is a texture that contains all the images specified in the `.texatlas` file. The `.texatlasidx` file stores coordinate and other image information.

**Adding Texture Atlases to a UI Canvas**

You can use the UI Editor to add texture atlases to a UI canvas.

**To add texture atlas references to a UI canvas**

1. Ensure that no elements are selected in the UI Editor and that the **Properties** pane displays the properties for **UI Canvas**.
2. Under **Editor settings**, note the **Texture atlases** property. The **Texture atlases** property lists the texture atlases that the UI canvas loads.

3. For **Texture atlases**, click the plus (+) sign to add one or more texture atlas child elements. One child element is required for each texture atlas to add to the UI canvas.
4. Click the **Browse (…)** button for an element that you added.

5. Choose the texture atlas for the element and then click **OK**.
Using Texture Atlases

The texture atlas that you added appears in the list.

6. To remove an element, click its (X) icon.

7. To remove all elements from the Texture atlases property, click the Remove all elements (☐) icon.

Using Texture Atlases to Reduce UI Draw Calls

The UI shader can combine draw calls that use up to a maximum of 16 textures. If this limit is exceeded, you can use texture atlases to reduce the number of draw calls.

Deciding which textures to add to a texture atlas is a two-part process:

1. Find out which canvases have draw calls that might be reduced by using texture atlases.
2. Determine which textures to place in specific texture atlases.
Using Texture Atlases

To gather data to make these determinations, you can use the Lumberyard Editor console.

To determine the textures to add to a texture atlas

1. In the Lumberyard Editor console, set the `ui_DisplayDrawCallData` console variable to `1` to display the draw call information for each loaded canvas, as in the following example.

   ```plaintext
   ui_DisplayDrawCallData 1
   ```

2. In the last column in the display of draw call information, note the values under `XTex`. The `XTex` column shows the number of draw calls when the shader supported maximum of 16 textures is reached. To reduce the number of draw calls, use a texture atlas.

3. In the Lumberyard Editor console, enter the `ui_ReportDrawCalls` console command. This command outputs a report of the draw calls for all active canvases into a text file.

4. Open the `lumberyard_version\dev\Cache\your_project_name\pc\user\log\LyShine\drawcallreport.txt` log file.

5. At the end of the report, examine the following two sections to identify the textures to put into texture atlases.

   ```plaintext
   Textures used on multiple canvases that are causing extra draw calls
   ```

   ```plaintext
   Per canvas report of textures used on only that canvas that are causing extra draw calls
   ```

6. When deciding which textures to put into a texture atlas, consider the following points:

   - A texture atlas reduces draw calls only if the textures in the atlas are used on screen at the same time. Therefore, to reduce draw calls, put textures that are going to be on screen at the same time into the same texture atlas.

   For example, suppose you have texture sets A and B. The textures in A appear in one screen state, and the textures in B appear in a different screen state. In this situation, put the textures in A into one texture atlas and the textures in B into a different texture atlas.

   - The default maximum size of a texture atlas is `4096x4096`.

   Because this limits the number of textures in a texture atlas, having the UI load multiple separate texture atlases is a good practice.

For more information on the `ui_DisplayDrawCallData` and `ui_ReportDrawCalls` commands, see [Debugging UI Canvases](p. 2906).
Debugging UI Canvases

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the following console commands and console variables to display debug information for the UI when your game is running.

- **ui_DisplayCanvasData** (p. 2906)
- **ui_DisplayDrawCallData** (p. 2907)
- **ui_DisplayElemBounds** (p. 2908)
- **ui_DisplayTextureData** (p. 2909)
- **ui_ReportDrawCalls** (p. 2909)

### ui_DisplayCanvasData

Displays canvas data for enabled or loaded canvases.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Off.</td>
</tr>
<tr>
<td>1</td>
<td>Displays a line of information for each loaded canvas.</td>
</tr>
<tr>
<td>2</td>
<td>Displays information only for canvases that are enabled.</td>
</tr>
</tbody>
</table>

The following example displays data for five loaded UI canvases.

<table>
<thead>
<tr>
<th>NN</th>
<th>Name</th>
<th>En</th>
<th>Po</th>
<th>Na</th>
<th>DO</th>
<th>nElem</th>
<th>nGrab</th>
<th>nMenu</th>
<th>nTitle</th>
<th>nText</th>
<th>nMask</th>
<th>nInput</th>
<th>nNotify</th>
<th>nActive</th>
<th>nHover</th>
<th>nClick</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Connects</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Connects</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>6</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Other</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>10</td>
<td>14</td>
<td>12</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>Mask</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>21</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>None</td>
</tr>
<tr>
<td>4</td>
<td>NotedMask</td>
<td>Y</td>
<td>Y</td>
<td>E</td>
<td>28</td>
<td>26</td>
<td>26</td>
<td>24</td>
<td>5</td>
<td>15</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>None</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>69</td>
<td>100</td>
<td>90</td>
<td>89</td>
<td>15</td>
<td>53</td>
<td>17</td>
<td>0</td>
<td>36</td>
<td>15</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

The following table describes each column.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>The index number of the canvas in the list. The canvases are listed in the order that they're drawn in.</td>
</tr>
<tr>
<td>Name</td>
<td>The leaf canvas name.</td>
</tr>
<tr>
<td>En</td>
<td>Whether the canvas is enabled.</td>
</tr>
<tr>
<td>Po</td>
<td>Whether the canvas accepts positional inputs (for example, mouse input).</td>
</tr>
<tr>
<td>Na</td>
<td>Whether the canvas has navigation enabled.</td>
</tr>
<tr>
<td>DO</td>
<td>The draw order, which is used to sort the list of loaded canvases.</td>
</tr>
<tr>
<td>nElem</td>
<td>The number of UI elements in the canvas.</td>
</tr>
</tbody>
</table>
### Debugging UI Canvases

#### Column Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nEnab</td>
<td>The number of enabled UI elements in the canvas. If a parent isn't enabled, the element isn't counted.</td>
</tr>
<tr>
<td>nRend</td>
<td>The number of enabled renderable elements in the canvas (how many images, text and particle effects are being rendered).</td>
</tr>
<tr>
<td>nRCtrl</td>
<td>The number of enabled &quot;render control&quot; elements in THE canvas (masks and faders).</td>
</tr>
<tr>
<td>nImg</td>
<td>The number of enabled UI elements with UiImageComponents.</td>
</tr>
<tr>
<td>nText</td>
<td>The number of enabled UI elements with UiTextComponents.</td>
</tr>
<tr>
<td>nMask</td>
<td>The number of enabled UI elements with UiMaskComponents.</td>
</tr>
<tr>
<td>nFadr</td>
<td>The number of enabled UI elements with UiFaderComponents.</td>
</tr>
<tr>
<td>nIntr</td>
<td>The number of enabled UI elements with interactable components (Button, Slider, TextInput, and so on).</td>
</tr>
<tr>
<td>nUpdt</td>
<td>Number of enabled UI elements with components that listen for updates (that is, that potentially do something every frame).</td>
</tr>
<tr>
<td>ActiveInt</td>
<td>The name of the active interactable on this canvas (if any).</td>
</tr>
<tr>
<td>HoverInt</td>
<td>The name of the current hover interactable on this canvas (if any).</td>
</tr>
</tbody>
</table>

#### ui_DisplayDrawCallData

Displays the number of draw calls used to render UI canvases. This variable is useful for performance tuning and debugging.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Turns off the display.</td>
</tr>
<tr>
<td>1</td>
<td>Turns on the display.</td>
</tr>
</tbody>
</table>

The following example data shows draw call information for four UI canvases.

<table>
<thead>
<tr>
<th>NN</th>
<th>Canvas name</th>
<th>nDraw</th>
<th>nPrim</th>
<th>nTris</th>
<th>nRTs</th>
<th>nUTex</th>
<th>nMask</th>
<th>XRT</th>
<th>XEInc</th>
<th>XSRgb</th>
<th>XPexV</th>
<th>XTex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mainmenu</td>
<td>1</td>
<td>12</td>
<td>254</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Components</td>
<td>1</td>
<td>14</td>
<td>326</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Dynamic</td>
<td>1</td>
<td>16</td>
<td>196</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>ManyChildren</td>
<td>5</td>
<td>50</td>
<td>656</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>6</td>
<td>68</td>
<td>1426</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The following table describes each column.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>The index number of the canvas in the list. The canvases are listed in the order that they're drawn in.</td>
</tr>
<tr>
<td>Name</td>
<td>The leaf canvas name.</td>
</tr>
</tbody>
</table>
### _Debugging UI Canvases_

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDraw</td>
<td>The number of draw calls.</td>
</tr>
<tr>
<td>nPrim</td>
<td>The number of primitives (for example, images and text strings).</td>
</tr>
<tr>
<td>nTris</td>
<td>The number of triangles rendered for the UI.</td>
</tr>
<tr>
<td>nMask</td>
<td>The number of mask render nodes in the render graph.</td>
</tr>
<tr>
<td>nRTs</td>
<td>The number of render target render nodes in the render graph.</td>
</tr>
<tr>
<td>nUTex</td>
<td>The number of unique textures being rendered in the canvas in this frame.</td>
</tr>
<tr>
<td>XMask</td>
<td>The number of draw calls caused by the use of masks. <strong>Note</strong> One mask can cause up to four extra draw calls.</td>
</tr>
<tr>
<td>XRT</td>
<td>The number of draw calls caused by render targets.</td>
</tr>
<tr>
<td>XBlnd</td>
<td>The number of draw calls caused by a change in blend mode.</td>
</tr>
<tr>
<td>XSrgb</td>
<td>The number of draw calls caused by a change in Srgb write. This data point appears only for render targets (for example, playing video).</td>
</tr>
<tr>
<td>XMaxV</td>
<td>The number of draw calls caused by a render node that requires more than 65536 vertices or 16384 quads. This case is uncommon. For example, more than 16000 characters of text would be required for a value to appear.</td>
</tr>
<tr>
<td>XTex</td>
<td>The number of draw calls that occur when the shader supported maximum of 16 textures is reached. To reduce the number of these calls, you can use a texture atlas. For more information, see <em>Using Texture Atlases</em> (p. 2900).</td>
</tr>
</tbody>
</table>

**ui_DisplayElemBounds**

This console command displays an overlay on the screen that shows the rectangles of elements. By default, it shows the rectangular bounds of every UI element for each enabled UI canvas.

If you have several enabled UI canvases and want to see the rectangular bounds for only one canvas, use the `ui_DisplayElemBoundsCanvasIndex` console variable. To use the `ui_DisplayElemBoundsCanvasIndex` console variable, specify the index of the canvas whose bounds you want to display. To find the index for an enabled canvas, use the `ui_DisplayCanvasData 2` setting.

The following example shows the rectangular bounds of nested scrollboxes.
ui_DisplayTextureData

Displays the textures that the UI is using.

The display shows the dimensions, data size, texture format, and pathname of each texture that the UI system is using in the current frame. The textures are sorted in descending order by the amount of memory that they use.

The following example shows data for 13 unique textures in the current frame.

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Data Size</th>
<th>Format</th>
<th>Texture Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 x 256</td>
<td>133072 B</td>
<td>RGBA</td>
<td>AB_button_2B</td>
</tr>
<tr>
<td>136 x 113</td>
<td>25000 B</td>
<td>RGBA</td>
<td>lyshoneexamples/outline.dos</td>
</tr>
<tr>
<td>38 x 38</td>
<td>577 b</td>
<td>RGBA</td>
<td>ui/textures/lyshoneexamples/panelbgd.dos</td>
</tr>
<tr>
<td>38 x 38</td>
<td>577 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/button normal.dds</td>
</tr>
<tr>
<td>38 x 38</td>
<td>577 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/dropdown_button.dds</td>
</tr>
<tr>
<td>64 x 64</td>
<td>1536 b</td>
<td>RGBA</td>
<td>ui/textures/lyshoneexamples/Button.dds</td>
</tr>
<tr>
<td>36 x 36</td>
<td>1536 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/dropdown_arrow.dds</td>
</tr>
<tr>
<td>24 x 24</td>
<td>2909 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/dropdown_arrow.dds</td>
</tr>
<tr>
<td>24 x 16</td>
<td>1536 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/slider_fill_normal.dds</td>
</tr>
<tr>
<td>24 x 16</td>
<td>1536 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/slider_track_normal.dds</td>
</tr>
<tr>
<td>32 x 32</td>
<td>1909 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/radiobutton_background_normal.dds</td>
</tr>
<tr>
<td>19 x 16</td>
<td>1296 b</td>
<td>RGBA</td>
<td>ui/textures/prefab/cellbutton_over.dds</td>
</tr>
<tr>
<td>8 x 8</td>
<td>8 b</td>
<td>RGBA</td>
<td>lyshoneexamples/Textures/White.dds</td>
</tr>
</tbody>
</table>

ui_ReportDrawCalls

Writes a report of draw calls to a log file.

The command output displays the location of the log file, as in the following example.

```
ui_ReportDrawCalls (Command)
ui ReportDrawCalls
[CONSOLE] Executing console command 'ui_ReportDrawCalls' (UI) - Wrote Draw Call Report to @log@/LyShine/DrawCallReport.txt
```
Defining Game and Level Load Screens

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can create a game or level loading screen with the UI Editor. The game loading screen is displayed while the game loads. The level loading screen is displayed while a level loads. You can create and define a loading screen for each level.

Load screens were refactored in Lumberyard v1.24 to support more than just UI canvases when using RAD Game Tools' Bink video files. You can specify a path to either a UI canvas or a Bink video file. Additionally, multi-threaded load screens are supported for Bink videos, enabling a load screen to render hitch-free while levels load.

To define the game and level loading screens, you set the file paths as parameters in game.cfg and level.cfg.

Defining a Game Loading Screen

To define a game loading screen, first do one of the following:

• Create the loading screen canvas in the UI Editor and save it in your Lumberyard project directory.
• Save a Bink video file in your Lumberyard project directory.

You then add or modify parameters in game.cfg, which is at the root of your project directory.

To add game loading screen parameters to game.cfg

1. In a text editor, open game.cfg at the root of your project directory.
2. Add or modify the following parameters in game.cfg:

   • game_load_screen_uicanvas_path – File path to the .uicanvas game load screen file relative to your project path. Use this if you are using a UI canvas for your loading screen.
   
   Example

   If your game load canvas is at lumberyard_version\dev\SamplesProject\UI\Canvases\UiAnimMultiSequence.uicanvas, you would specify the following path:

   \UI\Canvases\UiAnimMultiSequence.uicanvas

   • game_load_screen_bink_path – File path to the .bk2 game load screen file relative to your project path. Use this if you are using a Bink video for your loading screen.

   • game_load_screen_minimum_time – Minimum amount of time to show the game load screen, in seconds. Important to prevent short loads from flashing the load screen. 0 means there is no minimum. The default is 0.
Defining a Level Loading Screen

To define a level loading screen, first do one of the following:

- Create the loading screen canvas in the UI Editor and save it in your level's directory.
- Save a Bink video file in your level's directory.

You then add or modify parameters in level.cfg, which is at the root of your level directory.

To add level loading screen parameters to level.cfg

1. In a text editor, open level.cfg at the root of your level directory.
2. Add or modify the following parameters in level.cfg:

   - **level_load_screen_uicanvas_path** – File path to the .uicanvas level load screen file relative to your project path. Use this if you are using a UI canvas for your loading screen.

   **Example**

   If your level load canvas is at \lumberyard_version\dev\StarterGame\Levels\StarterGame\UiAnimMultiSequence.uicanvas (root of the level directory, which is the same as level.cfg), specify the following path:

   Levels\StarterGame\UiAnimMultiSequence.uicanvas

   - **level_load_screen_bink_path** – File path to the .bk2 level load screen file relative to your project path. Use this if you are using a Bink video for your loading screen.

   - **level_load_screen_minimum_time** – Minimum amount of time to show the level load screen, in seconds. Important to prevent short loads from flashing the load screen. 0 means there is no minimum. The default is 0.

   - **level_load_screen_sequence_to_auto_play** – The name of the level load screen animation sequence to play on load.

   - **level_load_screen_sequence_fix_fps** – A fixed frame rate for the level load screen animation to play on load. Default is 60. To ignore this setting and use the real time-delta, specify -1.
• ly_EnableLoadingThread – Experimental. Set to 1 to enable fully threaded loading where the load screen is drawn on a thread that isn't loading data. Currently only supported for Bink load screens.

The following are examples of these parameters in a level.cfg file:

```plaintext
level_load_screen_uicanvas_path="Levels\StarterGame\UiAnimMultiSequence.uicanvas"
level_load_screen_minimum_time=3
level_load_screen_sequence_to_auto_play="TopRowMove"
level_load_screen_sequence_fix_fps=4.0

level_load_screen_bink_path="Videos\IntroLevelLoadingScreen.bk2"
level_load_screen_minimum_time=3
```

## Working with UI Slices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The UI system uses the slices technology, which is a cascaded data management system for entities. This slice technology is similar to what the main Lumberyard Editor uses, but the user interface and some of the requirements are slightly different for the UI system.

For more information about component entity system slices, see Working with Slices (p. 510).

### Topics

- Creating a UI Slice (p. 2912)
- Instantiating a UI Slice (p. 2913)
- Creating a Cascaded UI Slice (p. 2913)
- Creating a Detached UI Slice (p. 2914)
- Modifying a UI Slice and Pushing Changes (p. 2914)
- Editing Slices in a New Tab (p. 2915)
- Flagging Dynamic Slices (p. 2916)

## Creating a UI Slice

A slice can contain any number of UI elements. However, all UI elements in a slice must be contained within one parent element.

You don't need to select all of the elements to go into the new slice. If you just select one element, all of its child elements are included in the slice.

**To create a UI slice**

1. In the **UI Editor**, in the **Hierarchy** pane or in the viewport, select the top-level parent entity to include in the slice.
Instantiating a UI Slice

You can create an instance of a slice in your UI canvas.

To instantiate a slice

In the UI Editor, do one of the following:

1. Choose New, Element from Slice Browser. This displays a file browser to select your slice file.
2. Choose New, Element from Slice Library. This displays a hierarchical menu of all the slices under \UI\Slices\Library.

Creating a Cascaded UI Slice

A cascaded slice is a slice that contains instances of other slices. Cascaded slices can store their own overrides for component and entity properties and can also contain their own entities.

When you create a cascaded slice, the UI system maintains references to the slices that are child entities. This means, for example, that you can instantiate an image slice as a parent and a text slice as a child. You can then select them both and make a cascaded slice called Button. If you then create instances of the Button slice, they all contain a reference to the image slice and the text slice. If you then push a font change to the text slice, this affects all instances of the Button slice as well as any other instances of the text slice.
To create a cascaded slice

1. Select the root of a set of elements. The child entities within that root can be individual elements, slices, or a combination of both.

   **Note**
   If your root is not already in a slice, and you want to maintain child slice references, you must also select the child slices in this step. If you select only the root, only one option appears: **Make New Slice from Selection**. This creates a detached UI slice (flattens child references).

2. Right-click the selection and then choose **Make Cascaded Slice from Selected Slices & Entities**.

3. Save the cascaded slice with a descriptive name.

Creating a Detached UI Slice

You can create a detached UI slice from an existing UI slice instance. When you create a detached UI slice, the UI system removes, or flattens, all references to child slices. Using the button example from the previous section, let's say you saved the image slice instance and its child text slice instance as a detached slice and called it `Button2`. The detached `Button2` would not reference any other slices; it is a slice that contains two entities. If you pushed a change to the text slice, it would not affect the text within any instances of the `Button2` slice.

To create a detached UI slice

1. Select the root of a set of elements, or a root and one or more child entities. The child entities within that root can be individual elements, slices, or a combination of both.

2. Right-click the selection and choose **Make Detached Slice from Selected Entities**.

   **Note**
   If the any of the elements that you selected are slice instances, two options appear: **Make Detached Slice from Selected Entities** and **Make Cascaded Slice from Selected Slices & Entities**.

3. Save the slice with a descriptive name.

Modifying a UI Slice and Pushing Changes

In the **UI Editor**, you modify UI slices and push changes similarly to the way that you do in the main Lumberyard Editor. For more information on pushing changes in Lumberyard's component entity system, see **Working with Slices** (p. 510).

The following conventions apply when working with slices in the **UI Editor**:

- If you select an entity that is part of a slice instance, the **Properties** pane highlights in orange any properties that are different between the selected entity and the slice to which it belongs.
- A UI slice can't contain references to any entities not within the slice. A UI slice can only contain reference to entities within the slice.

In the **Hierarchy** pane, elements that are part of a slice appear in blue. Elements that appear in bold blue indicate that the element is the root of the slice. Within the slice's root, elements that appear in italic indicate a child slice.

**Example**

- The element `Background` is not in a slice.
• **FontRenderingButton** is a root element and its child element, **Text**, is an element within submenubutton.slice.

You can locally change an instantiated slice's name, which obscures the name of the slice from which it instantiated.

**To determine the slice that the element instantiated from**

• Pause on the element. The slice name and its path appears in a tooltip.

**To push local changes to the slice**

1. Right-click in the **Properties** pane or on selected UI elements in the **Hierarchy** pane.
2. In the context menu, choose **Push to Slice**.

**Editing Slices in a New Tab**

When you edit an instance of a UI slice in a UI canvas, you can use the context menu to **Push to Slice**. This pushes the local changes that you select. For simplicity, you can instead make changes to the slice in its own context. To do this, you can create a blank UI canvas and instantiate the slice there to edit. The **UI Editor** automates this process with the **Edit slice in new tab** feature.

**To edit a slice in a new tab**

1. In the **Hierarchy** pane, right-click a slice element.
2. Choose **Edit slice in new tab** and then choose the slice you want to edit. Multiple choices appear only if the selected element is an instance of a cascaded slice.
The selected slice appears in a new tab labeled **Slice: slice name**.

3. Edit the slice in the new tab.

   **Note**
   
   If you add elements to the slice, the elements must be children of the slice instance. Any elements outside of the slice instance will not be saved, as this is a temporary canvas for editing the slice.

4. When finished, choose **File, Save Slice** to save your changes to the slice.

## Flagging Dynamic Slices

A slice flagged as a dynamic slice can be used like any other slice, but it can also be instantiated at runtime.

You can flag a slice as dynamic in the **Asset Browser**.

**To flag a slice as dynamic**

- Right-click the `.slice` file and choose **Set Dynamic Slice**.

To instantiate a UI slice at runtime, use the **UiSpawner** component on a UI element. This causes the system to automatically spawn a dynamic slice on activation. It also exposes a bus to Lua and C++ that allows the slice to be instantiated whenever needed.

## UI Elements

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the [AWS Game Tech blog](https://aws.amazon.com), to learn more.
UI elements are entities that you can attach multiple components to. You can start with an empty element and add components to it, such as a button, image, slider, text, and so on. Or you can add an existing pre-fabricated (prefab) element (p. 2920), such as a scroll box, which is an element with components already attached. You can also create your own prefab elements (p. 2920).

Every UI element has the following two required components that are added automatically:

- The **Element** component identifies the element **Id** and presents the option to start in a disabled state.

  The **Start enabled** option is selected by default. To start the element as disabled in the game, clear this option. When starting the element as disabled in the game, the player can't see or interact with that element or its children. This setting has no effect on how the element appears in the **UI Editor**'s viewport.

- The **Transform2D** component defines the positioning, spacing, and size of the element relative to its parent (whether its parent is the canvas or another element). For more information, see **Transform2D Component** (p. 2926).

When you select the **Editor Only** option, that element and its children exist only in the context of the **UI Editor**. For example, when the launcher, the game, or **Preview** mode in the **UI Editor** displays a UI canvas, elements marked **Editor Only** and its children don't appear.
You can use **Editor Only** elements when you author a UI canvas and you want to display additional info or visuals that appear only when editing. For example, you can add a UI mockup image of the intended final canvas and mark it as **Editor Only** so that it's visible only when editing the canvas. Those authoring the canvas can use the image as a guide to placing and configuring elements.

**Note**
Runtime references to the **Editor Only** element, such as from script or code, are invalid. You can reference **Editor Only** elements from other **Editor Only** elements. References to **Editor Only** elements from elements that aren't **Editor Only** cause an error and possibly undefined behavior.

Each UI element can also have one visual component (p. 2936) (image or text), one interactive component (p. 2947) (button, check box, scroll box, slider, or text input), and one layout component (p. 2972) (layout column, layout row, or layout grid). The remaining components (p. 2994) are the mask and fader, either or both of which UI elements can attach.

For each of the following procedures, use the **UI Editor** to manage UI elements.

### Managing UI Elements in the UI Editor

<table>
<thead>
<tr>
<th>Task</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>To find an element</td>
<td>In the <strong>UI Editor</strong> menu, choose <strong>Edit</strong> and then <strong>Find Elements</strong> or press <strong>Ctrl+F</strong>. For more information, see the section called “Searching for UI Elements” (p. 2921).</td>
</tr>
<tr>
<td>To create an element</td>
<td>In the <strong>UI Editor</strong> toolbar, click <strong>New</strong> and then <strong>Empty element</strong>. The element appears in the <strong>Hierarchy</strong> pane and viewport.</td>
</tr>
<tr>
<td>To move, rotate, or resize an element</td>
<td>Select the element and then click the <strong>Move</strong>, <strong>Rotate</strong>, or <strong>Resize</strong> tool in the toolbar. Select <strong>Snap to grid</strong> to modify elements in increments.</td>
</tr>
<tr>
<td>To copy an element</td>
<td>Right-click the element in the <strong>Hierarchy</strong> pane or viewport and choose <strong>Copy</strong>.</td>
</tr>
<tr>
<td>To nudge an element</td>
<td>To nudge, or move, an element one pixel at a time, select the element and click the <strong>Move</strong> tool. Use arrow keys to nudge elements in the selected direction. Press and hold the <strong>Shift</strong> key while pressing the arrow keys to nudge elements 10 pixels at a time.</td>
</tr>
<tr>
<td>To paste a copied element</td>
<td>Right-click anywhere in the <strong>Hierarchy</strong> pane or viewport and choose <strong>Paste</strong>. If an element is selected, the <strong>Paste as sibling</strong> and <strong>Paste as child</strong> options appear.</td>
</tr>
<tr>
<td>To delete an element</td>
<td>Right-click the element in the <strong>Hierarchy</strong> pane or viewport and choose <strong>Delete</strong>.</td>
</tr>
<tr>
<td>To hide an element</td>
<td>Click the eye icon (to the right of the element name) in the <strong>Hierarchy</strong> pane or viewport. Click again to show the element. When hiding or unhiding (showing) an element, the following behaviors apply:</td>
</tr>
<tr>
<td></td>
<td>• If you hide an element, that element and all of its descendants are hidden in the viewport.</td>
</tr>
<tr>
<td></td>
<td>• If all of the element's parents or ancestors are showing (not hidden), clicking that element's eye icon toggles only that element's visibility.</td>
</tr>
</tbody>
</table>
### Configuring UI Anchors and Offsets

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Each UI element's position is determined by the `Transform2D` component. The `Transform2D` component sets a UI element's position and size relative to its parent's edges. The parent may be another element (if the elements are nested), or the canvas.

For more information about the `Transform2D` component, see Managing UI Anchors and Offsets (p. 2926).

### Scaling to Device Resolutions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use the **Scale to Device** property on the **Transform2D** component with the anchoring system to build UIs that display on multiple screen resolutions. The **Scale to Device** property applies an element scale that's based on the ratio of the authored canvas size to the target canvas size.

To view the effects of this computation, open the **Preview** canvas in the UI Editor.

For more information about the **Transform2D** component's **Scale to Device** property, see **Scale to Device (p. 2930)**.

**Using and Creating UI Prefabs**

This system has been deprecated and will be removed in a future release.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In the **UI Editor**, prefabs are preconfigured UI elements and compound elements that you can add to a canvas. You can also create custom prefabs.

**To add a prefab element**

1. In the **UI Editor** toolbar, click **New, Element from prefab**.
2. Select from:
   - Button
   - Checkbox
   - Image
   - LayoutColumn
   - LayoutGrid
   - LayoutRow
   - ScrollBox
   - ScrollBarHorizontal
   - ScrollBarVertical
   - Slider
   - Text
   - TextInput

   The new element appears in the **Hierarchy** pane and viewport.

If you have created your own element or modified an existing prefab, you can save it as a custom prefab.

**To save a custom prefab element**

1. In the **UI Editor**, right-click an element that you have created or modified in the **Hierarchy** pane or viewport.
2. Click **(Deprecated) Save as Prefab**.
3. In the **Save As** dialog box, do the following:
a. Navigate to any location in the project folder where you want to save your prefab.
b. Name your prefab with a `.uiprefab` file extension.
c. Click Save.

Your prefab now appears in the **New, (Deprecated) Element from prefab** menu.

## Searching for UI Elements

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can find UI elements by filtering by their name or their attached components. The **Find Element** tool hierarchically displays UI elements for the active UI canvas.

To open the **Find Element** tool

- In the **UI Editor**, do one of the following:
  - From the menu, choose **Edit** and then **Find Elements**.
  - Press **Ctrl+F**.

Initially, all UI elements in the currently active UI canvas appear in a tree view.
Searching for UI Elements

To filter by UI element name

- Enter the name in the search filter box.

  UI element names that contain the entered text remain visible. All other UI elements are hidden.

To filter by component

1. Select the filter button next to the search filter box.
2. In the menu that appears, select one or more components.

  UI elements that contain the selected components are visible. All other UI elements are hidden.

  **Note**
  The Find Element tool displays for selection only the components used by the UI elements in the active UI canvas.

UI elements that don't match the search criteria are typically hidden. However, a parent element that doesn't match the search criteria remains visible if one of its children meets the search criteria. This maintains the tree structure used to display the UI elements. Displayed UI elements that meet search criteria appear as white text. Displayed UI elements that don't meet the criteria but are visible to maintain tree structure appear in gray text.
After you select one or more UI elements, you can choose to select them in your UI Editor hierarchy.

To transfer your selections to the Hierarchy panel

1. In the Find Elements tool, select one or more UI elements. Press Ctrl to select multiple elements.
2. Choose Select in Hierarchy.

The Hierarchy panel appears, with the same UI elements selected.

UI Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

UI components define the properties of a UI element. For example, every element has a Transform2D component that defines its position, rotation, size, and scale. You can give an element additional properties by adding components, such as adding the image component to give an element color or texture. Each UI element can have one visual component (image or text), one interactive component (button, check box, scroll box, slider, or text input), and one layout component (layout column, layout row, or layout grid). You can attach any or all of the remaining components: dynamic components, tooltips, tooltips display, mask, and fader.

Topics
- Working with UI Components (p. 2924)
- Transform2D Component (p. 2926)
- Visual Components (p. 2936)
- Interactive Components (p. 2947)
- Layout Components (p. 2972)
- Tooltip Components (p. 2985)
• Dynamic Components (p. 2987)
• UISpawner Component (p. 2990)
• Other Components (p. 2994)

Working with UI Components

You can use the UI Editor to add and delete components, create references to UI elements for component properties, and manage components.

Topics
• Adding or Deleting Components (p. 2924)
• Referencing UI Elements (p. 2924)
• Managing UI Components (p. 2925)

Adding or Deleting Components

You can add or delete components in the UI Editor (p. 2874).

To add a component to an element

1. In the UI Editor, select an element in the Hierarchy pane or in the Viewport.
   
   To add components to multiple elements at once, use Ctrl to select multiple elements.
2. Do one of the following:
   • At the top of the Properties panel, click Add Component.
   • In the Properties panel, right-click and choose Add Component.
3. Select the component to add to the element: image, text, button, check box, slider, text input, scroll box, fader, mask, layout column, layout row, or layout grid.
4. See the instructions for the specific component that you're adding.

To delete a component from an element

1. In the UI Editor, select an element in the Hierarchy pane.
2. In the Properties panel, right-click the component and click Remove.

Referencing UI Elements

You can specify a UI element for some properties on components. For example, you can specify UI elements for Lua Script properties.
Working with UI Components

You can specify elements as properties either by dragging the element into the property field or by using the pick object button.

**To drag a UI element into a property**
- In the **UI Editor**'s **Hierarchy** panel, select an element and drag it to the component property in the **Properties** panel.

**To use the pick object button to reference a UI element**
1. In the **Properties** panel, next to the property to specify an element for, click the pick object button.
2. Select the element that you want from either the viewport or the **Hierarchy** panel.
   The property populates with the UI element name.

**To clear the element name from the property**
- In the **Properties** panel, next to the property name, click x.

After you specify a UI element as a property, you can select that element from the **Properties** panel.

**To select a UI element from the Properties panel**
- In the **Properties** panel, double-click the element name.

**Managing UI Components**

You can use the context menu in the **Properties** panel to add, remove, cut, copy, and paste components.

**To manage components from the context menu**
1. In the **Hierarchy** panel, select the UI element.
2. In the **Properties** panel, either right-click the component or click the menu button in the header. Then choose one of the following:
   - **Add component** – Add a component to the element
   - **Delete component** – Remove a component from the element
   - **Cut component** – Cut a component from one UI element to be pasted onto a different UI element
   - **Copy component** – Copy a component from one UI element to be pasted onto a different UI element
• **Paste component** – Paste a component copied from one UI element onto a different UI element

You can also use the context menu to act on multiple components at once.

**To perform actions on multiple components**

1. Press **Ctrl** to select multiple components and then right-click to open the context menu.
2. Choose an action.

**Note**

- The **Element** and **Transform2D** components are automatically added to a UI element and can't be removed from the component list.

- Some actions are disabled, depending on the context. For example, you can't paste a component if you haven't copied one.

**Transform2D Component**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Transform2D** component is automatically added to every UI element you create. This component defines the element's position, rotation, size, and scale.

**Topics**

- Managing UI Anchors and Offsets (p. 2926)
- Scale to Device (p. 2930)

**Managing UI Anchors and Offsets**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use anchors and offset settings in the **Transform2D** component to set a UI element's position and size relative to its parent's edges. The **Transform2D** component is a required component in every element.

Anchor values are always 0.00% to 100.00% as defined by the parent's edges. Offsets are expressed in pixels and are relative to the anchors.

Anchors and offsets are useful in a variety of situations:

- Ensuring that an element maintains a specific padding within its parent's edges, regardless of changes to the parent's size
- Anchoring an element to a corner of its parent, regardless of changes to the parent's size or position
- Building resolution-independent UI elements
For example, you can ensure an element remains full screen regardless of the screen's resolution.

**To configure an element's anchors**

1. In the **Hierarchy** pane of the **UI Editor** (p. 2874), select the element whose anchors you want to modify.
2. In the **Properties** pane, under **Transform2D**, choose from the selection of commonly used anchor placements.
   1. Anchor to the parent's center, corner, or midway along an edge without changing size.
   2. Anchor to the left edge, middle, or right edge; vertical size adjusts to parent.
   3. Anchor to the top edge, middle, or bottom edge; horizontal size adjusts to parent.
   4. Anchor all of the element's edges to the parent. The horizontal and vertical size adjusts to parent. You can use this anchor preset to place an element that remains full screen, regardless of a change in resolution. This applies if the canvas is its parent.

**To further edit (fine tune) an element's anchors**

In the **Properties** pane, under **Transform2D**, do the following for **Anchors**, as appropriate:

- For **Left**, enter a value between 0.00% and 100.00%.
- For **Right**, enter a value between 0.00% and 100.00%.
- For **Top**, enter a value between 0.00% and 100.00%.
- For **Bottom**, enter a value between 0.00% and 100.00%.

The anchors' positions can be visualized as points on a grid, plotted in percentages by the length of its parent's edges from left to right and top to bottom. If you want to keep the element's size absolute (so that it doesn't change size when the parent changes size) but want to anchor it a particular vertical or horizontal point relative to the parent's size, make sure the top and bottom (or left and right) anchors have the same number. In this case, the anchors are said to be together.
But if, for example, you want the element's left and right edges to each remain at a fixed percentage relative to its parent and to change size as its parent changes size, then make the numbers different. In this case, the anchors are called split.

To edit an element's position and size

In the Properties pane, under Transform2D, modify the Offsets, as appropriate:

If the element's anchors are together, do the following:
- For X Pos, enter a negative or positive value in pixels. This adjusts the horizontal offset relative to the left-right anchor position.
- For Y Pos, enter a negative or positive value in pixels. This adjusts the vertical offset relative to the top-bottom anchor position.

When the element's anchors are together, only its position adjusts with the parent's size. The element's size is not adjusted. Therefore, you can manually adjust the element's size, which remains consistent when anchors are together.
- For Width, enter a value in pixels.
- For Height, enter a value in pixels.

If the element's anchors are split, do the following:
- For Left, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's left anchor.
- For Right, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's right anchor.
- For Top, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's top anchor.
- For Bottom, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's bottom anchor.
To edit an element's pivot, rotation, and scale

In the Properties pane, under Transform2D, do the following for Pivot, Rotation, and Scale, as appropriate:

- For Pivot, select a pivot preset or enter values for X and Y where 0 and 1 represent the element's edges.
- For Rotation, enter a value in degrees.
- For X Scale, enter a value to use as a multiplier for the element's width.
- For Y Scale, enter a value to use as a multiplier for the element's height.
- Select Scale to Device if you want the UI element and its child elements to scale with the device resolution.

Note
The element rotates around, resizes from, and calculates position from its pivot point. The pivot point is not limited by the element's borders. You can place the pivot outside of the element.

Example: Using Anchors to Resize an Element Relative to its Parent

In the following example, anchors are used to resize the element relative to its parent. The layout column of buttons is resized as needed to stay on the screen. Because the layout column of buttons does not use the Scale to Device setting, the button text does not change size along with its parent button.

Note
You can configure the text element (p. 2943) separately.

The layout column of buttons has the following settings.

<table>
<thead>
<tr>
<th>Layout Column element settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Property</strong></td>
</tr>
<tr>
<td>Anchors</td>
</tr>
<tr>
<td>Pivot</td>
</tr>
<tr>
<td>Scale to device</td>
</tr>
</tbody>
</table>
Scale to Device

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Scale to Device property helps build game UIs that can display on multiple screen resolutions. You can preview your canvas at different resolutions in the UI Editor in Preview Mode.

A device scale is computed by using the ratio of the authored canvas size to the runtime canvas size. The device scale is then adjusted based on the selected Scale to Device setting. When you select any Scale to Device setting other than None, the device scale is multiplied with the Transform2D component's Scale property to get the final local scale for the element.

The following Scale to Device settings are available:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Does not scale with the device resolution.</td>
</tr>
<tr>
<td>Scale to fit (uniformly)</td>
<td>Scales to fit while maintaining the aspect ratio.</td>
</tr>
<tr>
<td></td>
<td>The final device scale for both X and Y is the minimum of the width and height ratios between the authored canvas size to the viewport size.</td>
</tr>
<tr>
<td>Scale to fill (uniformly)</td>
<td>Scales to fill while maintaining the aspect ratio.</td>
</tr>
<tr>
<td></td>
<td>The final device scale for both X and Y is the maximum of the width and height ratios between the authored canvas size to the viewport size.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Scale to fit X (uniformly)</td>
<td>Scales to fit horizontally while maintaining the aspect ratio. The final device scale for both X and Y is the ratio between the authored canvas width to the viewport width.</td>
</tr>
<tr>
<td>Scale to fit Y (uniformly)</td>
<td>Scales to fit vertically while maintaining the aspect ratio. The final device scale for both X and Y is the ratio between the authored canvas height to the viewport height.</td>
</tr>
<tr>
<td>Stretch to fill (non-uniformly)</td>
<td>Stretches to fill horizontally and vertically without maintaining the aspect ratio. The final device scale is the ratio between the authored canvas size to the viewport size.</td>
</tr>
<tr>
<td>Stretch to fit X (non-uniformly)</td>
<td>Stretches to fit horizontally, but doesn't stretch vertically. The final device scale for X is the ratio between the authored canvas width to the viewport width. Y doesn't scale with the device resolution.</td>
</tr>
<tr>
<td>Stretch to fit Y (non-uniformly)</td>
<td>Stretches to fit vertically, but doesn't stretch horizontally. The final device scale for Y is the ratio between the authored canvas height to the viewport height. X doesn't scale with the device resolution.</td>
</tr>
</tbody>
</table>

When using the *Scale to Device* setting, note the following:

- Scaling is performed about the element's pivot.
- Scaling an element doesn't affect the value of its offsets from its anchors.
- The element's final scale includes any scale inherited from its parents. Set the *Scale to Device* property on a UI element whose child elements that you also want to scale with the device resolution.
- Setting *Scale to Device* on a UI element and a descendant element results in double scaling on the descendant element.
- Avoid setting *Scale to Device* on a UI element that doesn't have its anchors together. Doing so can result in undesired behavior. This is because the anchors affect the size of the element relative to its parent, and the *Scale to Device* scale is applied on top of that.

**Example**

The element's size matches the viewport's size if you set the anchors to the following values:

- Left = 0%
- Top = 0%
- Right = 100%
- Bottom = 100%

However, if you then add a scale on top of these anchor values, the element size no longer matches the viewport size.
Scale to Device Examples

Each of the following examples demonstrates a different Scale to Device setting.

In each example, the background image covers the whole screen and uses the following settings:

- Anchors (apart): Left = 0%, Top = 0%, Right = 100%, Bottom = 100%
- Image type: Tiled
- Scale to Device: None

Uniform Scaling

In this uniform scaling example, the UI parent element has a fixed aspect ratio and is centered and fitted to the screen that it's displayed on.

The background image is a texture with simple settings (p. 2932).

The elements that make up the UI are all contained in a parent element and has the following settings.

Parent UI element settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchor</td>
<td>Left = 50%, Top = 50%, Right = 50%, Bottom = 50%</td>
</tr>
<tr>
<td>Pivot</td>
<td>Default settings: X = 0.5, Y = 0.5</td>
</tr>
<tr>
<td>Width and Height</td>
<td>Matches the authored canvas size (for example, 1280x720)</td>
</tr>
<tr>
<td>Scale to device</td>
<td>Scale to fit (uniformly)</td>
</tr>
</tbody>
</table>
Scale to Fit Y

In this uniform scale to fit Y example, the layout column and its buttons are uniformly scaled so that they fit vertically on the screen regardless of its resolution.

The background image is a texture with simple settings (p. 2932).

The elements that make up the buttons are contained in a layout column element. The layout column element contains the UI buttons and has the following settings.

### Layout Column element settings for uniform scaling to fit Y

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchors</td>
<td>Left = 50%, Top = 50%, Right = 50%, Bottom = 50%</td>
</tr>
<tr>
<td>Pivot</td>
<td>Default settings: X = 0.5, Y = 0.5</td>
</tr>
<tr>
<td>Scale to device</td>
<td>Scale to fit Y (uniformly)</td>
</tr>
</tbody>
</table>

Uniform Scaling While Maintaining Relative Position

In this example, the **Scale to Device** setting scales the health bar and speed indicator depending on the screen resolution. Anchor settings maintain their positions so that the health bar always appears in the right corner and the speed indicator always appears in the center.

The background image is a texture with simple settings (p. 2932).

The health bar element has the following settings. The anchor values keep it on the upper right corner of the screen.
Health bar element settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchors</td>
<td>Left = 100%, Top = 0%, Right = 100%, Bottom = 0%</td>
</tr>
<tr>
<td>Pivot</td>
<td>X = 1.0, Y = 0.0</td>
</tr>
<tr>
<td>Scale to device</td>
<td>Scale to fit (uniformly)</td>
</tr>
</tbody>
</table>

The speed indicator element has the following settings. The anchor values keep it in the top center of the screen.

Health bar element settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchors</td>
<td>Left = 50%, Top = 0%, Right = 50%, Bottom = 0%</td>
</tr>
<tr>
<td>Pivot</td>
<td>X = 0.5, Y = 0.0</td>
</tr>
<tr>
<td>Scale to device</td>
<td>Scale to fit (uniformly)</td>
</tr>
</tbody>
</table>

The following images show how the health bar and speed indicator scales based on the screen resolution while maintaining their position on the screen.

**Note**

The indicated resolutions are not shown to scale.
Visual Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add one visual component to an element: Image, Particle Emitter, or Text.

Topics
- Image (p. 2936)
- Particle Emitter (p. 2938)
- Text (p. 2943)

Image

You can use an Image component to add a color tint or texture to an element.

To see in-game examples of completed canvases with Image components

1. In Lumberyard Editor, in the Samples Project (p. 146), open the UiFeatures level.
2. Press Ctrl+G to play the game and then choose fComponents, and then Visual Components, and then Image.

You can view examples of image types such as sliced, stretched, fixed, tiled, stretched to fit, and stretched to fill.
3. Press Esc to exit the game.

To view these same canvases in the UI Editor, navigate to the lumberyard_version\dev\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Image directory.

You can open the following canvases:
- ColorTest.uicanvas
- ImageTypes.uicanvas

Use the Properties pane of the UI Editor to configure the following settings for the Image component.

Image Settings

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpriteType</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td>Sprite/Texture asset</td>
<td>Image displays the asset specified for Sprite path</td>
</tr>
<tr>
<td>Render target</td>
<td>Image displays the render target specified in Render target name</td>
</tr>
<tr>
<td>Sprite path</td>
<td>Click the browse (...) icon and select a suitable file.</td>
</tr>
<tr>
<td></td>
<td>Click the open-in (arrow) icon next to Sprite path to open the Sprite Editor (p. 2882).</td>
</tr>
<tr>
<td>Render target name</td>
<td>Enter a name of a render target and press Enter.</td>
</tr>
</tbody>
</table>
## Visual Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The sprite sheet image index that the component will render.</td>
</tr>
<tr>
<td>Color</td>
<td>Click the color swatch to select a different color.</td>
</tr>
<tr>
<td></td>
<td>Displays only if the SpriteType is Sprite/Texture asset and the image has</td>
</tr>
<tr>
<td></td>
<td>been configured as a sprite sheet using the Sprite Editor (p. 2882).</td>
</tr>
<tr>
<td>Alpha</td>
<td>Use the slider to choose an alpha value between 0 and 1.</td>
</tr>
<tr>
<td>Image type</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Stretched – Stretches the texture with the element without maintaining</td>
</tr>
<tr>
<td></td>
<td>aspect ratio</td>
</tr>
<tr>
<td></td>
<td>• Sliced – Treats the texture as a 9-sliced sprite</td>
</tr>
<tr>
<td></td>
<td>• Fixed – Makes the texture pixel perfect</td>
</tr>
<tr>
<td></td>
<td>• Tiled – Tiles the texture to fill the element</td>
</tr>
<tr>
<td></td>
<td>• Stretched to Fit – Scales to fit while maintaining aspect ratio</td>
</tr>
<tr>
<td></td>
<td>• Stretched to Fill – Scales to fill while maintaining aspect ratio</td>
</tr>
<tr>
<td>Blend Mode</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Normal – Uses alpha to interpolate colors between elements</td>
</tr>
<tr>
<td></td>
<td>• Add – Blends colors between elements by adding (lightening) color values</td>
</tr>
<tr>
<td></td>
<td>together</td>
</tr>
<tr>
<td></td>
<td>• Screen – Blends colors using inverse source color resulting in a lighter</td>
</tr>
<tr>
<td></td>
<td>color</td>
</tr>
<tr>
<td></td>
<td>• Darken – Chooses the darker color channel when blending between elements</td>
</tr>
<tr>
<td></td>
<td>• Lighten – Chooses the lighter color channel value when blending between</td>
</tr>
<tr>
<td></td>
<td>elements</td>
</tr>
<tr>
<td>Fill Type</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• None – Displays the entire image</td>
</tr>
<tr>
<td></td>
<td>• Linear – Uses Fill Amount value to fill the image linearly from one edge</td>
</tr>
<tr>
<td></td>
<td>to the opposite edge</td>
</tr>
<tr>
<td></td>
<td>• Radial – Uses Fill Amount value to fill the image radially 360 degrees</td>
</tr>
<tr>
<td></td>
<td>around the center of the image</td>
</tr>
<tr>
<td></td>
<td>• Radial Corner – Uses Fill Amount value to fill the image radially 90</td>
</tr>
<tr>
<td></td>
<td>degrees around a corner of the image</td>
</tr>
<tr>
<td></td>
<td>• Radial Edge – Uses Fill Amount value to fill the image radially 180</td>
</tr>
<tr>
<td></td>
<td>degrees around the midpoint of an edge of the image</td>
</tr>
<tr>
<td>Fill Amount</td>
<td>The amount of the image to be filled, from 0.00 to 1.00.</td>
</tr>
<tr>
<td>Fill Start Angle</td>
<td>The start angle for the fill, measured in degrees clockwise from vertical.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Corner Fill Origin</td>
<td>The starting corner that the image is filled from. Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Top Left</td>
</tr>
<tr>
<td></td>
<td>• Top Right</td>
</tr>
<tr>
<td></td>
<td>• Bottom Left</td>
</tr>
<tr>
<td></td>
<td>• Bottom Right</td>
</tr>
<tr>
<td>Edge Fill Origin</td>
<td>The edge from which the image is filled about (radial corner) or from (linear). Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Left</td>
</tr>
<tr>
<td></td>
<td>• Top</td>
</tr>
<tr>
<td></td>
<td>• Right</td>
</tr>
<tr>
<td></td>
<td>• Bottom</td>
</tr>
<tr>
<td>Fill Clockwise</td>
<td>If selected, the image is radially filled clockwise about the fill center.</td>
</tr>
<tr>
<td>Fill Center</td>
<td>If selected, the center segment of a slice-resized sprite is visible.</td>
</tr>
</tbody>
</table>

**Particle Emitter**

You can use a **Particle Emitter** component to emit two-dimensional particles from an element.

**To see in-game examples of completed canvases with a Particle Emitter component**

1. In Lumberyard Editor, in the **Samples Project** (p. 146), open the UiFeatures level.
2. Press **Ctrl+G** to play the game and then choose in order **Components**, **Visual Components**, and **Particle Emitter**.

   You can view particle emitter examples that create a variety of spark effects and trail effects.
3. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `lumberyard_version\dev\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\ParticleEmitter` directory.

You can open the following canvases:

- **ParticleEmitter.uicanvas**
- **ParticleEmitterSparks.uicanvas**
- **ParticleEmitterTrails.uicanvas**

Use the **Properties** pane of the **UI Editor** to configure the following settings for the **Particle Emitter** component.
Emitters Settings

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emit on activate</td>
<td>Start emitting when the component is activated.</td>
</tr>
<tr>
<td>Hit particle count on activate</td>
<td>Emit and process the average number of particles when the emitter starts emitting.</td>
</tr>
<tr>
<td>Infinite life time</td>
<td>Make the lifetime of the emitter infinite.</td>
</tr>
<tr>
<td>Emitter life time</td>
<td>Enter the number of seconds that the emitter is active during. When the end of the lifetime is reached, the emitter stops emitting. This option is available when Infinite life time isn't set.</td>
</tr>
<tr>
<td>Emit rate</td>
<td>Enter the number of particles to emit per second.</td>
</tr>
<tr>
<td>Emitter shape</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Point – Particles are emitted from the emitter position</td>
</tr>
<tr>
<td></td>
<td>• Circle – Particles are emitted from within the circle that is contained by the element area</td>
</tr>
<tr>
<td></td>
<td>• Quad – Particles are emitted from within the quad that is contained by the element area</td>
</tr>
<tr>
<td>Particle count limit</td>
<td>Use the Active particle limit value to limit the number of active particles.</td>
</tr>
<tr>
<td>Active particles limit</td>
<td>This option is available when Particle Count Limit is set. Type the maximum number of active particles. When the maximum number is reached, additional particles are emitted only after existing particles are removed. The maximum value is 9999.</td>
</tr>
<tr>
<td>Fixed random seed</td>
<td>Specify a fixed random seed for the emitter. When not selected, a random seed is generated each time the emitter starts emitting.</td>
</tr>
<tr>
<td>Random seed</td>
<td>This option is available when Fixed random seed is set. Enter the numerical seed to use for the particle emitter when Fixed random seed is selected. The field accepts a maximum nine-digit negative or positive integer.</td>
</tr>
<tr>
<td>Emit on edge</td>
<td>This option is available when Emitter shape is Circle or Quad. Emit particles on the edge of the specified shape.</td>
</tr>
<tr>
<td>Emit inside distance</td>
<td>This option is available when Emit on Edge is set. Enter the distance inside the edge that the particles are emitted from.</td>
</tr>
<tr>
<td>Emit outside distance</td>
<td>Type the distance outside the edge that the particles are emitted from.</td>
</tr>
<tr>
<td>Initial direction type</td>
<td>This option is available when Emitter shape is Circle or Quad. Select one of the following to specify how the initial direction is calculated:</td>
</tr>
<tr>
<td></td>
<td>• Relative to emit angle – Use the value of emit angle as the direction along which to emit</td>
</tr>
<tr>
<td></td>
<td>• Relative to emitter center – Use the direction from the emitter center to the initial position as the direction to emit along</td>
</tr>
<tr>
<td>Emit angle</td>
<td>Enter the number of degrees vertical that particles are emitted from.</td>
</tr>
</tbody>
</table>
### Visual Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emit angle variation</td>
<td>Enter a number or use the slider to specify, in degrees, the variation of the emit angle. Valid values are from 0 through 180. A value of 10 specifies a variation range of plus or minus 10 degrees.</td>
</tr>
</tbody>
</table>

### Particle Settings

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite life time</td>
<td>Make the particle lifetime infinite.</td>
</tr>
<tr>
<td>Life time</td>
<td>Enter the number of seconds that the emitted particles are initially active during.</td>
</tr>
<tr>
<td>Life time variation</td>
<td>Enter the number of seconds that the lifetime of the emitted particles can vary.</td>
</tr>
<tr>
<td>Sprite pathname</td>
<td>Click the ellipsis (…) to open the Pick Texture dialog box and select a sprite image file.</td>
</tr>
<tr>
<td>Animated sprite sheet</td>
<td>This option is available when the selected sprite is a sprite sheet (has more than one cell). Select to change the sprite sheet cell index on each particle over time.</td>
</tr>
<tr>
<td>Loop sprite sheet animation</td>
<td>This option is available when Animated sprite sheet is set. Select to loop the sprite sheet cell animation.</td>
</tr>
<tr>
<td>Random sprite sheet index</td>
<td>This option is available when the selected sprite is a sprite sheet (has more than one cell). Select to randomly choose the initial sprite sheet cell index.</td>
</tr>
<tr>
<td>Sprite sheet index</td>
<td>This option is available when the selected sprite is a sprite sheet (has more than one cell) and Random sprite sheet index isn’t set. Select the sprite sheet index that is used for the emitted particles.</td>
</tr>
<tr>
<td>Sprite sheet start frame</td>
<td>This option is available when the selected sprite is a sprite sheet (has more than one cell) and Random sprite sheet index is set. Select the starting frame of the sprite sheet range for sprite sheet animation or randomly choosing the index.</td>
</tr>
<tr>
<td>Sprite sheet end frame</td>
<td>This option is available when the selected sprite is a sprite sheet (has more than one cell) and Random sprite sheet index is set. Sets the end frame of the sprite sheet range for sprite sheet animation or randomly choosing the index.</td>
</tr>
<tr>
<td>Sprite sheet frame delay</td>
<td>This option is available when Animated sprite sheet is set. Type the number of seconds of delay between each sprite sheet frame.</td>
</tr>
<tr>
<td>Blend mode</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Normal – Use alpha to interpolate colors between elements.</td>
</tr>
<tr>
<td></td>
<td>• Add – Blend colors between elements by adding (lightening) color values together.</td>
</tr>
<tr>
<td></td>
<td>• Screen – Use inverse source color to blend colors. Results in a lighter color.</td>
</tr>
<tr>
<td></td>
<td>• Darken – Choose the darker color channel value when blending between elements.</td>
</tr>
</tbody>
</table>
### Visual Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lighten</strong></td>
<td>Choose the lighter color channel value when blending between elements.</td>
</tr>
</tbody>
</table>

### Particle Movement

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative to emitter</td>
<td>Move particles relative to the element to which the Particle Emitter component is attached. When this option isn't selected, the element leaves a trail of particles as the emitter moves around the canvas.</td>
</tr>
</tbody>
</table>
| Movement co-ordinate type | Select the type of coordinate space for particle movement:  
- Cartesian  
- Polar |
| Speed | This option is available when Movement co-ordinate type is set to Cartesian. Enter a number that specifies the initial speed of the emitted particles when the emit direction is calculated. |
| Speed variation | This option is available when Movement co-ordinate type is set to Cartesian. Enter a number that specifies the variation in initial speed of the emitted particles when the emit direction is calculated. |
| Initial velocity | This option is available when Movement co-ordinate type is set to Polar. Enter X and Y values that specify the initial velocity of the emitted particles. |
| Initial velocity variation | This option is available when Movement co-ordinate type is set to Polar. Enter X and Y values that specify the variation in the initial velocity of the emitted particles. |
| Acceleration co-ordinate type | Select the type of coordinate space that is used for particle acceleration:  
- Cartesian  
- Polar |
<p>| Acceleration | Enter X and Y values that specify the acceleration of each emitted particle. |
| Orientation velocity based | Point the top of each particle toward the current velocity vector. |
| Initial orientation velocity based | Point the top of each particle toward the initial velocity vector. |
| Initial rotation | Enter the number of degrees of the initial rotation clockwise from vertical. |
| Initial rotation variation | Enter the number of degrees in the variation of the initial rotation. A value of 10 specifies a variation range of plus or minus 10 degrees around the specified initial rotation. |
| Rotation speed | Enter the rotation speed in degrees clockwise per second. |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation speed variation</td>
<td>Type the variation in rotation speed in degrees clockwise per second. A value of 10 specifies a variation range of plus or minus 10 degrees in the specified rotation speed.</td>
</tr>
<tr>
<td>Particle Size</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Lock aspect ratio</td>
<td>Locks the width and height of the emitted particles into the current aspect ratio.</td>
</tr>
<tr>
<td>Pivot</td>
<td>Enter X and Y values that specify the pivot for the particles from (0,0) at the top left to (1,1) at the bottom right.</td>
</tr>
<tr>
<td>Size</td>
<td>Enter X and Y values that specify the size of each emitted particle.</td>
</tr>
<tr>
<td>Size variation</td>
<td>Enter X and Y values that specify the variation in size of each emitted particle.</td>
</tr>
<tr>
<td>Particle Color</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Color</td>
<td>Enter RGB values that specify the color of each emitted particle or click the white square to use the Select Color dialog box.</td>
</tr>
<tr>
<td>Color brightness variation</td>
<td>Enter a decimal number between 0 and 1 that specifies the variation in brightness of each emitted particle.</td>
</tr>
<tr>
<td>Color tint variation</td>
<td>Enter a decimal number between 0 and 1 that specifies the tint variation of each emitted particle.</td>
</tr>
<tr>
<td>Alpha</td>
<td>Enter a decimal number between 0 and 1 that specifies the alpha that is used for the emitted particles.</td>
</tr>
<tr>
<td>Timelines</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Speed multiplier</td>
<td>Click the plus sign (+) to add keyframes that control a curve to multiply the particle speed over its lifetime.</td>
</tr>
<tr>
<td>Width multiplier</td>
<td>This option is available when Lock aspect ratio (in Particle Size) isn't set.</td>
</tr>
<tr>
<td></td>
<td>Click the plus sign (+) to add keyframes that control a curve to multiply the particle width over its lifetime.</td>
</tr>
<tr>
<td>Height multiplier</td>
<td>This option is available when Lock aspect ratio (in Particle Size) isn't set.</td>
</tr>
<tr>
<td></td>
<td>Click the plus sign (+) to add keyframes that control a curve to multiply the particle height over its lifetime.</td>
</tr>
<tr>
<td>Size multiplier</td>
<td>Click the plus sign (+) to add keyframes that control a curve to multiply the particle size over its lifetime.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Color multiplier</td>
<td>Click the plus sign (+) to add keyframes that control a curve to multiply the particle color over its lifetime.</td>
</tr>
<tr>
<td>Alpha multiplier</td>
<td>Click the plus sign (+) to add keyframes that control a curve to multiply the particle alpha over its lifetime.</td>
</tr>
<tr>
<td>Time</td>
<td>Enter a value between 0 and 1 that specifies when the keyframe occurs during the particle lifetime.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 is the start of the particle lifetime and 1 is the end of the particle lifetime.</td>
</tr>
<tr>
<td>Multiplier</td>
<td>Specify a value from -100 through 100 to multiply against the value that this timeline controls.</td>
</tr>
<tr>
<td></td>
<td>For example, if the speed value is 50.0 and the speed multiplier for a specific keyframe is 2.0, the value of speed at the specified keyframe is 100.</td>
</tr>
<tr>
<td>In tangent</td>
<td>Controls the in tangent for the current keyframe. Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Ease In</td>
</tr>
<tr>
<td></td>
<td>• Linear</td>
</tr>
<tr>
<td></td>
<td>• Step</td>
</tr>
<tr>
<td>Out tangent</td>
<td>Controls the out tangent for the current keyframe. Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Ease In</td>
</tr>
<tr>
<td></td>
<td>• Linear</td>
</tr>
<tr>
<td></td>
<td>• Step</td>
</tr>
<tr>
<td>Ease In</td>
<td>Specifies a gradual approach of a curve to a flat tangent.</td>
</tr>
<tr>
<td></td>
<td>For example, a keyframe that specifies Ease In for In tangent and Ease Out for Out tangent looks like the flattened tangent of an x^3 (x-cubed) curve at its origin.</td>
</tr>
<tr>
<td>Ease Out</td>
<td>Specifies a gradual retreat of a curve from a flat tangent.</td>
</tr>
<tr>
<td></td>
<td>For example, a keyframe that specifies Ease In for In tangent and Ease Out for Out tangent looks like the flattened tangent of an x^3 (x-cubed) curve at its origin.</td>
</tr>
<tr>
<td>Linear</td>
<td>Specifies that the curve moves linearly from the keyframe towards the next or previous keyframe.</td>
</tr>
<tr>
<td>Step</td>
<td>Specifies that the curve jumps from the current keyframe value to the next or previous keyframe value.</td>
</tr>
</tbody>
</table>

### Text

You can use a **Text** component to add a text string to an element.

#### To see in-game examples of completed canvases with text components

1. In Lumberyard Editor, in the **Samples Project** (p. 146), open the **UiFeatures level**.
2. Press **Ctrl+G** to play the game and then choose in order **Components, Visual Components**, and **Text**. The level has examples of text alignment, color and alpha, styling markup, overflow and wrapping, and character and line spacing.
3. Press Esc to exit the game.

To view the same canvases in the UI Editor, navigate to the `lumberyard_version\dev\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Text` directory.

You can open the following canvases:

- **Alignment.uicanvas** – Examples of aligning the text in its element rectangle
- **ColorAlpha.uicanvas** – Examples of setting different colors and transparency levels
- **StylingMarkup.uicanvas** – Examples of customizing the appearance of text using different styles, colors, and font in a single string
- **ImageMarkup.uicanvas** – Example of embedding images in text
- **OverflowWrap.uicanvas** – Examples of different wrap and overflow options

Use the **Properties** pane of the UI Editor to configure the following settings for the **Text** component.

**Text Settings**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Enter the preferred text string and press Enter. You can also apply text styling markup (p. 2945).</td>
</tr>
<tr>
<td>Enable markup</td>
<td>If selected, the text string is parsed for markup tags. For more information, see Text Markup (p. 2945).</td>
</tr>
<tr>
<td>Color</td>
<td>Click the color swatch to select a different color.</td>
</tr>
<tr>
<td>Alpha</td>
<td>Use the slider to choose an alpha value between 0 and 1.</td>
</tr>
<tr>
<td>Font path</td>
<td>Click the button and select a font .font file. For more information, see Adding New Fonts (p. 3000).</td>
</tr>
<tr>
<td>Font size</td>
<td>Enter a font size and press Enter.</td>
</tr>
<tr>
<td>Font effect</td>
<td>Select an effect from the list. The available font effects are dictated by the font .font file.</td>
</tr>
<tr>
<td>Horizontal text alignment</td>
<td>Select Left, Center, or Right to align the text with the element's left and right borders.</td>
</tr>
<tr>
<td>Vertical text alignment</td>
<td>Select Top, Center, or Bottom to align the text with the element's top and bottom borders.</td>
</tr>
<tr>
<td>Overflow mode</td>
<td>Select Overflow to allow the text to display beyond the edges of the element. Select Clip text to hide, or clip, any text that flows beyond the element's edges.</td>
</tr>
<tr>
<td>Wrap text</td>
<td>Select No wrap to prevent text from wrapping to subsequent lines. Select Wrap text to allow text to be broken into separate lines.</td>
</tr>
</tbody>
</table>
**Text Markup**

You can customize the appearance of the text in your game UI with bold and italic styling, multiple text colors, and multiple fonts in a single text string. You can also embed images in the text.

To do this, enter specific tags directly into the Text box, along with your string. The simple markup language is loosely based on HTML.

To use the text styling markup feature, you must use a font family `*.fontfamily` asset file in the Font path setting (rather than an individual `.font` asset file). For more information about adding font families to your projects, see Implementing New Fonts (p. 2999).

**To use text styling markup**

1. In the UI Editor (p. 2874), add a Text component to an element on your canvas (or modify an existing component).
2. With the element selected, in the Properties pane, set the Font path property to a `*.fontfamily` file.
3. Enter a string with markup styling in the Text box. See the next section for examples.

**Styling Tags and Attributes**

**Example**

You can use the following tags and attributes when styling text with markup:

**Bold** tag: `<b>`

![This text is bold]

**Italic** tag: `<i>`

![This text is italic]
**Font color** tag: `<font color>`

![Image showing text color change](image)

This text is red

**Font face** tag: `<font face>`

![Image showing font face change](image)

This text uses notosans font

---

**Image Tag and Attributes**

Use the `<img>` tag to embed an image in text.

**Example**

```html
<img src="images/icons/button" vAlign="center" height="fontHeight" scale="1.5" yOffset="5" xPadding="6"/>

<img src="images/icons/button" vAlign="bottom" height="100" scale="0.8" yOffset="-3" lPadding="2" rPadding="6"/>
```

Use the following attributes to customize your image. Only the `src` attribute is required. All other attributes are optional.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>src</code></td>
<td><code>src=&quot;images/icons/button&quot;</code></td>
<td>Asset path of the texture to display.</td>
</tr>
<tr>
<td><code>vAlign</code></td>
<td><code>vAlign=&quot;center&quot;</code></td>
<td>Alignment of image in the text area. The text area refers to the total height of the largest glyph in the font. If you don't specify this attribute, the default alignment is baseline.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• &quot;top&quot; – Aligns top of image to top of text area</td>
</tr>
</tbody>
</table>
## Interactive Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>height</strong></td>
<td>height=&quot;100&quot;</td>
<td>Sets the image height and adjusts its width to preserve the aspect ratio. If you don't specify this attribute, the default alignment is fontAscent.</td>
</tr>
<tr>
<td><strong>scale</strong></td>
<td>scale=&quot;0.8&quot;</td>
<td>Scales the image by the specified scale factor. If not specified, this attribute defaults to 1.0.</td>
</tr>
<tr>
<td><strong>yOffset</strong></td>
<td>yOffset=&quot;-3&quot;</td>
<td>Offsets the image vertically by the specified float value in pixels. If not specified, this attribute defaults to 0.</td>
</tr>
<tr>
<td><strong>xPadding</strong></td>
<td>xPadding=&quot;6&quot;</td>
<td>Adds equal spacing before and after the image by the specified float value in pixels. If not specified, this attribute defaults to 0.0.</td>
</tr>
<tr>
<td><strong>lPadding</strong></td>
<td>lPadding=&quot;10&quot;</td>
<td>Adds spacing to the left of the image by the specified float value in pixels. If not specified, this attribute defaults to 0.0.</td>
</tr>
<tr>
<td><strong>rPadding</strong></td>
<td>rPadding=&quot;2&quot;</td>
<td>Adds spacing to the right of the image by the specified float value in pixels. If not specified, this attribute defaults to 0.0.</td>
</tr>
</tbody>
</table>

- "center" – Aligns center of image to center of text area
- "bottom" – Aligns bottom of image to bottom of text area
- "baseline" – Aligns bottom of image to baseline of text area
- "fontHeight" – Sets image height to height of total text area
- "fontAscent" – Sets image height to the font's ascent
- "<float value>" – Sets image height to specified float value in pixels
Interactive components respond to user input. For example, the user can click a button or drag a slider. You can use Lua scripts (p. 2897) or Script Canvas graphs to link the component response to an action.

An interactive element is defined as an element that has an interactive component applied.

**Topics**
- Properties for Interactive Components (p. 2948)
- Button (p. 2953)
- Checkbox (p. 2954)
- RadioButton (p. 2955)
- RadioButtonGroup (p. 2956)
- Slider (p. 2957)
- TextInput (p. 2958)
- ScrollBar (p. 2960)
- ScrollBox (p. 2962)
- Draggable (p. 2966)
- DropTarget (p. 2968)
- Dropdown (p. 2969)
- DropdownOption (p. 2971)

**Properties for Interactive Components**

All of the interactive components share a common set of properties. These properties are grouped into the following categories:

- **Input Enabled** (p. 2949) – Check box or flag that determines whether the element can be interacted with.
- **Multi-Touch Input Enabled** (p. 2949) – Check box or flag that determines whether the element will handle multi-touch events. Requires multi-touch to be enabled on the parent UI canvas (enabled by default).
- **States** (p. 2950) – Settings that determine the appearance of the element when in the Hover, Pressed, or Disabled states.
- **Navigation** (p. 2951) – Settings that determine how the gamepad or arrow keys navigate between interactive elements (p. 2952).
- **Auto Activate** (p. 2952) – Check box or flag that determines whether the element is automatically activated when the player pauses on the element.
- **Actions** (p. 2953) – Events that are caused by the listed action.
Input Enabled

The **Input Enabled** setting, selected by default, determines whether the component can be interacted with.

To visualize how the interactive element looks in its disabled state, deselect the **Input Enabled** setting, and then use Preview mode (p. 2891) to preview your canvas.

Multi-Touch Input Enabled

The **Multi-Touch Input Enabled** check box is selected by default. This setting determines whether the component will handle all multi-touch events it receives. When the check box is unchecked, this interactable will handle only primary touch input events.

Lumberyard ignores this setting if **Handle multi-touch** is not enabled in the parent UI Canvas component.
States

The States group of properties defines the appearance of the interactive element and its child UI elements when the element is in the Hover, Pressed, and Disabled states.

The normal appearance of a visual element (defined as an element with a visual component, such as image or text) is defined by the properties of that visual component. Some of the visual component's properties, however, can be overridden by an interactive component that is in the Hover, Pressed, and Disabled states.

The Hover, Pressed, and Disabled states have a list of state actions, which define the appearance of that state, and which override the corresponding property of the visual component:

- **Color** – RGB color tint
- **Alpha** – Opacity
- **Sprite** – Texture
- **Font** – Text font and font effect (of a text component, for example)

The state actions—Color, Alpha, Sprite, and Font—each have a Target property that specifies which visual element is to be affected. The elements from which you can choose include the current element—listed as <This element>, its child elements, and the descendants of its child elements. Using the Target property, you can pick exactly which visual element to override.

For example, the button prefab (p. 2920) has a top element named Button that has a visual component to define its color. It also has a child element with a text component to define the text (and its color) of the button. The top element (Button) also has the Interactable component. The Target for the color state action can override either the Button element's color or the Text element's color, depending on what you select from the list.
When you first add an interactive component to an element, no state actions are added by default. You must add state actions to the states that you want to use and modify.

To add a state action to a state

In the UI Editor (p. 2874), in the Properties pane, under the interactive component's name (for example, Button), do the following:

1. Under Interactable, States, click Add new element (green +).
2. From the list, choose one of the following: Color, Font, Sprite, Alpha.

To delete a state action

- Click Remove element (red x) next to the state action that you want to delete.

To clear all state actions from a state

- Click Clear container (box icon) next to the state from which you want to clear all the state actions.

Navigation

You can use the Navigation group of properties to specify how the arrow keys or gamepad navigates between interactive elements.

For each interactive element, you can set navigation to one of the following:

- Automatic – Algorithm determines which interactive elements become focused when up, down, left, or right is pressed.
Interactive Components

- **Custom** – You manually specify the interactive elements that become focused when up, down, left, or right is pressed.
- **None** – This option removes navigation capability; using the keyboard or gamepad, the player cannot focus on this element.

**First Focus Element**

To determine which element receives first focus when a canvas is first loaded, set the **First Focus Element** in the Canvas Properties (p. 2894). The **First Focus Element** receives focus upon canvas load when no mouse is detected. If you do not set a **First Focus Element**, or if a mouse is detected, no element has focus until the user provides direction input from a keyboard or mouse. At that point, the element closest to the top left corner of the canvas receives focus.

Once an element has focus, navigation to other elements is controlled by the navigation properties (p. 2951) defined for each interactive component.

**Interactive Element Controls**

To interact with a focused element, press **Enter** on the keyboard.

When an element is interactive, use the following controls:

- **Button** and **Checkbox** – **Enter** presses the button or selects or clears the check box, and then returns to navigation automatically. The control does not remain active after action.

- **Slider** and **Scrollbox** – Use arrow keys or joystick to move the slider or scroll box. Press **Enter** to return to navigation.

- **TextInput** – While active, use the following (press **Enter** to return to navigation):
  - Arrow keys move the text cursor.
  - **Shift**+arrow keys selects text.
  - Alphanumeric keys type text at the cursor position.
  - **Ctrl+A** selects the entire text string.
  - **Backspace** deletes the character to the left of the cursor.
  - **Delete** deletes the character to the right of the cursor.

**Auto Activate**

The **Auto Activate** setting applies to interactive elements that remain active after they are pressed. These include the slider, scroll bar, scroll box, and text input.

When Auto Activate is set, an element automatically activates when navigated to with the directional keys, and deactivates when navigated away from with the directional keys. The user does not need to press the **Enter** and **Back** keys to activate or deactivate the element.
Interactive Components

The UI must be designed so that a user cannot get stuck on an activated element. For example, a horizontal slider that is set to automatically activate should not have another interactive element to the left or right of it. This is because pressing the left or right key on an active slider moves the slider handle.

Actions

You can use the Actions properties to trigger a particular event when one of the listed actions occur. Type a string for one of the actions. When the listed action occurs (for example, when a game player starts to pause on the element), the listed string is sent as an action.

You can enter strings for the following actions:

- Hover start
- Hover end
- Pressed
- Release

Button

You can use a Button component to make an element behave like a button.

To see an in-game example of a completed canvas with the Button component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Interactable Components, Button. You can view the different types of buttons you can create. Press Esc to exit the game.

To view this same canvas in the UI Editor, open \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Button\Styles.uicanvas.

Note the following:

- This component is typically applied to an element with an image component; if no visual or image component is present, many of the button's properties have no effect.

- If you want to add a text label to a button, add a child element with a text component.

- To define borders for a sliced image type, open the Sprite Editor. To do this, click the arrow (open-in) next to Sprite path.
You can add a prebuilt **Button** element from the slice library. When you do this, a basic button with the text string "Button" is automatically created in your **Hierarchy** pane.

**To add a Button element from the slice library**

- In the **UI Editor (p. 2874)**, choose **New, Element from Slice Library, Button**.

**To edit a button component**

In the **UI Editor (p. 2874) Properties** pane, expand **Button** and do the following, as appropriate:

  **Interactable**

  See **Properties (p. 2948)** to edit the common interactive component settings.

  **Actions, Click**

  Enter a text string. This string is sent as an action on the UI canvas when the button is clicked.

---

**Checkbox**

You can use the **Checkbox** component to make an element behave like a check box. This component is typically applied to an element with two visual child elements: One element appears when the check box is selected and another appears when the check box is cleared.

To see in-game examples of completed canvases with the **Checkbox** component, open the level **UiFeatures** in the project **SamplesProject**. Press **Ctrl+G** to play the game, and then choose **Components, Interactable Components, Checkbox**. You can view examples of color targets, check box interaction areas, and on and off elements. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\CheckBox` directory. You can open the following canvases:

- **Area.uicanvas** – Examples of different check box interaction areas
- **ColorTargets.uicanvas** – Examples of different color targets when interacting with the check box
- **OnOff.uicanvas** – Examples of different check box on and off elements

You can add a prebuilt **Checkbox** element from the slice library. When you do this, a basic check box with the text string "Checkbox" and a check image for the box is automatically created in your **Hierarchy** pane.

**To add a Checkbox element from the slice library**

- In the **UI Editor (p. 2874)**, choose **New, Element from Slice Library, Checkbox**.

**To edit a check box component**

In the **Properties** pane of the **UI Editor (p. 2874)**, expand **Checkbox** and do the following, as appropriate:

  **Interactable**

  See **Properties (p. 2948)** to edit the common interactive component settings.
Elements, On

Select an element from the list to specify the entity that appears when the check box state is on (selected).

Elements, Off

Select an element from the list to specify the entity that appears when the check box state is off (cleared).

Value, Checked

Select the box to change the initial state of the check box.

Actions, Change

Enter a text string. This string is sent as an action on the UI canvas when the check box has any state changes.

Actions, On

Enter a text string. This string is sent as an action on the UI canvas when the check box state changes to on (selected).

Actions, Off

Enter a text string. This string is sent as an action on the UI canvas when the check box state changes to off (cleared).

RadioButton

You can use the RadioButton component to make an element behave like a radio button. This component is typically used on an element with two visual child elements—one to display when the radio button is selected and another to display when the radio button is cleared.

Use this component in conjunction with the RadioButtonGroup component. The RadioButtonGroup component handles selecting and clearing the radio buttons in the group and makes sure that only one radio button is ever selected.

To see in-game examples of completed canvases with the RadioButton component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Interactable Components, RadioButton. You can view examples of different behaviors, default settings, and groups for radio buttons Press Esc to exit the game.

To view these same canvases in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\RadioButton directory. You can open the following canvases:

- Groups.uicanvas – Examples of different radio button groupings
- RadioButton.uicanvas – Examples of different behaviors and default settings

To edit a radio button component

In the Properties pane of the UI Editor (p. 2874), expand RadioButton and do the following, as appropriate:
Interactable

See Properties (p. 2948) to edit the common interactive component settings.

Elements, On

Select an element from the list to specify the entity that appears when the radio button state is on (selected).

Elements, Off

Select an element from the list to specify the entity that appears when the radio button state is off (cleared).

Elements, Group

Select an element from the list to specify the group the radio button belongs to.

Value, Checked

Select the box to change the initial state of the radio button.

Actions, Change

Enter a text string. This string is sent as an action on the UI canvas when the radio button has any state changes.

Actions, On

Enter a text string. This string is sent as an action on the UI canvas when the radio button state changes to on (selected).

Actions, Off

Enter a text string. This string is sent as an action on the UI canvas when the radio button state changes to off (cleared).

RadioButtonGroup

You can use the RadioButtonGroup component to manage radio buttons. This component handles selecting and clearing the radio buttons in the group when appropriate. It also ensures that only one radio button is selected at one time. You typically use this component on an element with children radio buttons, which are part of the radio button group.

To see in-game examples of completed canvases with the RadioButtonGroup component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Interactable Components, RadioButton. You can view examples of different behaviors, default settings, and groups for radio buttons Press Esc to exit the game.
To view these same canvases in the **UI Editor**, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\RadioButton directory. You can open the following canvases:

- Groups.uicanvas – Examples of different radio button groupings
- RadioButton.uicanvas – Examples of different behaviors and default settings

You can add a prebuilt **RadioButtonGroup** element from the slice library. When you do this, a group of three radio buttons is automatically created in your **Hierarchy** pane.

**To add a RadioButtonGroup element from the slice library**

- In the **UI Editor** (p. 2874), choose **New, Element from Slice Library, RadioButtonGroup**.

**To edit a RadioButtonGroup component**

In the **Properties** pane of the **UI Editor** (p. 2874), expand **RadioButtonGroup** and do the following, as appropriate:

**Settings, Allow uncheck**

Select to enable the clearing or unchecking of selected radio buttons.

**Actions, Change**

Enter a text string. This string is sent as an action on the UI canvas when the radio button group has any state changes.

---

**Slider**

You can use the **Slider** component to make an element behave like a slider. This component is typically applied to an element with three visual child elements: one immediate child, called **Track**, and two child elements of the track, called **Fill** and **Handle**.

To see in-game examples of completed canvases with the **Slider** component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose **Components, Interactable Components, Slider**. You can view examples of different slider behavior and positioning. Press Esc to exit the game.

To view these same canvases in the **UI Editor**, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Slider directory. You can open the following canvases:

- Behavior.uicanvas – Examples of slider behaviors such as fill type, handles, and buttons to control the slider
- Rotation.uicanvas – Example of slider positioning

You can add a prebuilt **Slider** element from the slice library. When you do this, the slider's track, fill, and handle are automatically created in your **Hierarchy** pane.

**To add a Slider element from the slice library**

- In the **UI Editor** (p. 2874), choose **New, Element from Slice Library, Slider**.
To edit a slider component

In the Properties pane of the UI Editor (p. 2874), expand Slider and do the following, as appropriate:

**Interactable**

See Properties (p. 2948) to edit the common interactive component settings.

**Elements, Track**

Select an element from the list to provide the background of the slider and to limit the movement of the manipulator.

**Elements, Fill**

Select an element from the list to provide the background of the slider, from the lower limit to the center of the manipulator position.

**Elements, Manipulator**

Select an element from the list to provide the movable knob of the slider.

**Value, Value**

Enter the initial value of the slider.

**Value, Min**

Enter the lower limit of the slider.

**Value, Max**

Enter the upper limit of the slider.

**Value, Stepping**

Enter the step value. For example, use 1 to only permit whole integer values.

**Actions, Change**

Enter a text string. This string is sent as an action on the UI canvas when the slider has finished changing values.

**Actions, End Change**

Enter a text string. This string is sent as an action on the UI canvas when the slider has changing values.

---

**TextInput**

You can use a TextInput component to make an element offer player input. This component is typically applied to an element with an image component and two child elements with text components (one for placeholder text and one for input text).

To see in-game examples of completed canvases with the TextInput component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components,
Interactable Components, TextInput. You can view examples of different types of text input behavior on single lines and on multiple lines. Press Esc to exit the game.

To view these same canvases in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\TextInput directory. You can open the following canvases:

- Multiline.uicanvas – Examples of editing multiline text strings
- SingleLine.uicanvas – Example of editing single-line text strings

You can add a prebuilt TextInput element from the slice library. When you do this, a text input box, pause-on state, and placeholder text "Type here..." are automatically created in your Hierarchy pane.

To add a TextInput element from the slice library

- In the UI Editor (p. 2874), choose New, Element from Slice Library, TextInput.

To edit a text input component

In the Properties pane of the UI Editor (p. 2874), expand TextInput and do the following, as appropriate:

Interactable

See Properties (p. 2948) to edit the common interactive component settings.

Elements, Text

Select an element from the list to provide the text component for the input text. The list shows child elements that have text components.

Elements, Placeholder text element

Select an element from the list to provide the text component for the placeholder text. The list shows child elements that have text components.

Text editing, Selection color

Click the color swatch to select a different color for the selected text.

Text editing, Cursor color

Click the color swatch to select a different color for the cursor.

Text editing, Max char count

Enter the maximum number of characters allowed in the text input box. Type -1 for no character limit.

Text editing, Cursor blink time

Enter a value in seconds. Use 0 for no blink, 1 to blink once every second, 2 to blink once every two seconds, etc.

Text editing, Is password field

Select the box and specify the replacement character.

Text editing, Clip input text

Sets the Overflow mode of the text element to Clip text at runtime.

Actions, Change

Enter a text string. This string is sent as an action on the UI canvas whenever a change occurs in the text input, such as typing or deleting a character.
Actions, End edit

Enter a text string. This string is sent as an action on the UI canvas whenever the player clicks off the text input or presses Enter.

Actions, Enter

Enter a text string. This string is sent as an action on the UI canvas when the player presses Enter.

ScrollBar

You can use a ScrollBar component to add a scrollable bar, or handle, for manipulating settings or scrolling within a scroll box.

To see in-game examples of completed canvases with the ScrollBar component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Interactable Components, ScrollBar. You can view examples of different types of scroll bar positioning and handles, scroll bars paired with scroll boxes, and visibility options. Press Esc to exit the game.

To view these same canvases in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\ScrollBar directory. You can open the following canvases:

- ScrollBoxes.uicanvas – Examples of scroll bars paired with scroll boxes
- Simple.uicanvas – Examples of scroll bar with simple logic
- Visibility.uicanvas – Examples of scroll bar visibility options when paired with scroll boxes

This is a horizontal scroll bar:

This is an image within a scroll box with both a horizontal and a vertical scroll bar:

In Lumberyard version 1.24 and later, the scroll bar can also fade automatically when not in use:
You can add a prefabricated horizontal or vertical scroll bar element. When you do this, a handle is automatically created and nested in your Hierarchy pane.

You can add a prebuilt ScrollBarHorizontal or ScrollBarVertical element from the slice library. When you do this, the scroll bar and its handle is automatically created in your Hierarchy pane.

**To add a ScrollBar element from the slice library**

- In the UI Editor (p. 2874), choose New, Element from Slice Library, ScrollBarHorizontal or ScrollBarVertical.

**To edit a scroll bar component**

In the Properties pane of the UI Editor (p. 2874), expand ScrollBar and do the following, as appropriate:

- **Interactable**

  See Properties (p. 2948) to edit the common interactive component settings.
Elements, Handle

Select an element from the list to provide the movable handle of the scroll bar.

Values, Orientation

Select the scroll bar's orientation:
- **Horizontal** – Scrollbar's handle moves left and right.
- **Vertical** – Scrollbar's handle moves up and down.

Values, Value

Enter the initial value of the scroll bar (0.0 to 1.0).

Values, Handle size

Enter the size of the handle relative to the scroll bar (0.0 to 1.0).

Values, Min handle size

Enter the minimum size of the handle in pixels.

Actions, Change

Enter a text string. This string is sent as an action on the UI canvas when the scroll bar changes values.

Actions, End Change

Enter a text string. This string is sent as an action on the UI canvas when the scroll bar has finished changing values.

Fade, Auto Fade When Not In Use (v1.24 and later)

Select the check box to enable the scrollbar to fade to transparency after it is not used for a set amount of time. Specify the delay time in **Fade Delay**.

Fade, Fade Delay (v1.24 and later)

Enter the delay in seconds before the scrollbar begins fading to transparency. Requires **Auto Fade When Not In Use** to be checked.

Fade, Fade Speed (v1.24 and later)

Enter the time in seconds that it will take for the scrollbar to completely fade to transparency. Requires **Auto Fade When Not In Use** to be checked.

**ScrollBox**

You can use a **ScrollBox** component to present content, such as images or text, within a scrollable area.

This component is typically used with a mask component, which hides the content outside of the masked area.
To see in-game examples of completed canvases with the `ScrollBox` component, open the level `UiFeatures` in the project `SamplesProject`. Press `Ctrl+G` to play the game, and then choose `Components, Interactable Components, ScrollBox`. You can view examples of different scrolling options, snapping options, interactions between scroll boxes and other components, and nested scroll boxes. Press `Esc` to exit the game.

To view these same canvases in the `UI Editor`, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\ScrollBox` directory. You can open the following canvases:

- `Interactions.uicanvas` – Examples of interactions between scroll boxes and other interactive components
- `Nested.uicanvas` – Examples of nesting scroll boxes
- `Scrolling.uicanvas` – Examples of different scrolling options such as horizontal, vertical, 2D, and unconstrained
- `Snapping.uicanvas` – Examples of different snapping options

You can add a prebuilt `ScrollBox` element from the slice library. When you do this, a mask, content, and image elements are automatically created and nested in your `Hierarchy` pane.

**To add a ScrollBox element from the slice library**

- In the `UI Editor` (p. 2874), choose `New, Element from Slice Library, Scrollbox`.

The element named `ScrollBox` (1) has the `ScrollBox` component (2) applied to it. You can add an image to the `ScrollBox` element's `Image` component (3), which acts as the visual frame for the scroll box. Because the mask element and its child elements are drawn in front of the scroll box element, you see only the edges of the image on the `ScrollBox` component. To increase or decrease the viewable area of this image, adjust the offsets in the mask element's `Transform2D` (p. 2926) component.
The element named Mask has a Mask component applied to it, which acts as the viewport through which you can see the content. To specify a custom mask, you can add an image to the Mask element’s Image component. The contents are drawn to the visible area of the mask; the transparent area of the mask hides content.

To edit a scroll box component

In the Properties pane of the UI Editor, expand ScrollBox and do the following, as appropriate:

**Interactable**

See Properties to edit the common interactive component settings.

**Content, Content element**

Select an element from the list to provide the content to be displayed within the scroll box.

**Content, Initial scroll offset**

Enter the initial offset value of the content element's pivot point from the parent element's pivot point.

**Content, Constrain scrolling**

Select the check box to prevent content from scrolling beyond its edges.

**Content, Snap**

Select a snapping mode:

- **None** – No snapping.
- **To children** – When a drag motion is released, the content element moves in such a way that the closest child element's pivot point is snapped to the parent element's pivot point. You can use this, for example, to center a child element in the scroll box when the dragging stops.
• **To grid** – When a drag motion is released, the content element's pivot point is snapped to a multiple of the grid spacing from the parent element's pivot point.

**Horizontal scrolling, Enabled**

Select the check box to enable content to scroll horizontally. If the element, or its parent, is rotated, then the axis of scrolling is also rotated. You can enable horizontal scrolling simultaneously with vertical scrolling to scroll in both directions.

**Horizontal scrolling, Scrollbar element**

Select an element from the list to provide the horizontal scroll bar associated with the scroll box.

**Horizontal scrolling, Scrollbar visibility**

Select the visibility behavior of the horizontal scroll bar:

- **Always visible** – Scroll bar is always visible.
- **Auto hide** – Scroll bar is automatically hidden when not needed. Scroll bar is resized according to visibility of the vertical scroll bar.
- **Auto hide and resize view area** – Same as Auto hide, but the view area is also resized smaller when the scroll bar is visible and larger when the scroll bar is hidden.

**Vertical scrolling, Enabled**

Select the check box to enable content to scroll vertically. If the element, or its parent, is rotated, then the axis of scrolling is also rotated. You can enable vertical scrolling simultaneously with horizontal scrolling to scroll in both directions.

**Vertical scrolling, Scrollbar element**

Select an element from the list to provide the vertical scroll bar associated with the scroll box.

**Vertical scrolling, Scrollbar visibility**

Select the visibility behavior of the vertical scroll bar:

- **Always visible** – Scroll bar is always visible.
- **Auto hide** – Scroll bar is automatically hidden when not needed. Scroll bar is resized according to visibility of the vertical scroll bar.
- **Auto hide and resize view area** – Same as auto hide, but the view area is also resized smaller when the scroll bar is visible and larger when the scroll bar is hidden.

**Actions, Change**

Set the action that is triggered during dragging each time the position changes.

**Actions, End change**

Set the action that is triggered when a drag motion is completed.
Draggable

You can use the Draggable component to make a UI element movable from one location to another on screen. Use the Draggable component in combination with the DropTarget component so that dragging can start on the draggable element and end on the drop target element. Drag-and-drop is a common operation in UI screens, such as an inventory system.

Because drag-and-drop behavior is game specific, the Draggable and DropTarget components are designed to be used with scripting or C++ to define actions that result from the drag and the drop.

To add the draggable component to a UI element, use the Add Component menu in the Properties pane.

The following picture shows an example of a Draggable component, where color has been added to the state actions for Drag States.
The **Draggable** component is an interactive component. It has the standard interactive properties (p. 2948).

**To edit a Draggable component**

In the **Properties** pane of the **UI Editor** (p. 2874), expand **Draggable** and do the following, as appropriate:

**Interactable**

See **Properties** (p. 2948) to edit the common interactive component settings.

**Drag States**

Define the color, alpha, sprite, or font of this element and its child elements for a particular drag state.

When it is not being dragged, an element with a draggable component uses the interactable states (pause on, pressed, and disabled).

When being dragged, however, the draggable component has an additional three states:

- **Normal** – Automatic state when a drag state begins.

- **Valid** – Typically the state used when the draggable component pauses on a valid drop target. This state is determined by a script that you write or C++ code that connects to the **UiDropTargetNotificationBus** and listens for the **OnDropHoverStart** method.
**Invalid** — Typically the state used when a draggable component pauses over an invalid drop target. This state is determined by a script that you write or C++ code. When a valid drag state is triggered, a notification is automatically sent using the `UiDropTargetNotificationBus`.

The script or C++ can use the `UiDraggableBus` to set the drag state of the `Draggable` component. It can also set the drop state of the `DropTarget` to indicate valid drop targets to the user.

To see an example of simple drag Lua script, open `DraggableElement.lua` in `LyShineExamples\Assets\UI\Scripts\LyShineExamples\DragAndDrop`.

---

**DropTarget**

You can use the `DropTarget` component to implement drag-and-drop behavior with the `Draggable` component.

Because drag-and-drop behavior is game specific, the `Draggable` and `DropTarget` components are designed to be used with scripting or C++ to define actions that result from the drag and the drop.

To add the `DropTarget` component to a UI element, use the Add Component menu in the Properties pane.

The following picture shows an example of a `DropTarget` component, where color has been added to the state actions for Drop States.

The `DropTarget` component shares properties with interactive components, such as state actions and navigation settings.

**To edit a DropTarget component**

In the Properties pane of the UI Editor (p. 2874), expand `DropTarget` and do the following, as appropriate:

**Drop States**

Define the color, alpha, sprite, or font of this element and its child elements in a valid or invalid drop state. By default, the drop state of a drop target is normal, which means that there are no visual overrides.
During a drop, the drop target component can be **Valid** or **Invalid**. Because the drop target component has no knowledge of what is a valid drag-and-drop operation, you use a script or C++ to switch the drop target into the **Normal**, **Valid**, and **Invalid** states. This is usually accomplished by connecting to the `UiDropTargetNotifications` bus and listening for the `OnDropHoverStart` and `OnDropHoverEnd` notifications.

**Navigation**

Navigation settings control how keyboard or gamepad navigation works during a drag-and-drop operation. When using the keyboard, you can press **Enter** on a draggable element to enter drag mode. Then you can use the arrow keys to move the element from one drop target to another using the navigation settings specified here.

**Actions, On Drop**

Enter a text string. This string is sent as an action on the UI canvas whenever a draggable is dropped on this drop target. For better control, we recommend that you use the `UiDropTargetNotifications` bus instead.

To see an example of simple drop target Lua script, open `DropTarget.lua` in Gems \LyShineExamples\Assets\UI\Scripts\LyShineExamples\DragAndDrop.

**Dropdown**

You can use the **Dropdown** component to make an element behave like a drop-down menu. Use this component with child elements that contain content. The child elements provide the contents of the drop-down menu.

You can use the **Dropdown** component with the **DropdownOption** component. With the **DropdownOption** component, you can configure options to change the menu text and its icon after it is selected.

To see in-game examples of completed canvases with the **Dropdown** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **Dropdown**. You can view **Simple Dropdowns**, **Nested Dropdowns**, **Multiple Select & Functionality**, and **Using UI Components and Dropdowns**. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Dropdown directory. You can open the following canvases:

- **MultipleFunc.uicanvas** – Multiple selection drop-down menu and functional drop-down menu (perform actions on a ball such as create, destroy, move, and change color)
- **Nested.uicanvas** – Two levels and multilevel with siblings submenus
- **Simple.uicanvas** – Simple selection drop-down menu, selection drop-down menu with icons, expand on pause drop-down menu, and expand on pause with icons
- **UsingUi.uicanvas** – Drop-down menus with a scroll box, image, check box, slider, and radio buttons

You can add a prebuilt **Dropdown** element from the slice library. When you do this, a drop-down menu, three options, and their image elements are automatically created in your **Hierarchy** pane.
To add a Dropdown element from the slice library

- In the UI Editor (p. 2874), choose New, Element from Slice Library, Dropdown.

To edit a Dropdown component

In the Properties pane of the UI Editor (p. 2874), expand Dropdown and do the following, as appropriate:

**Interactable**

See Properties (p. 2948) to edit the common interactive component settings.

**Elements, Content**

Select an element from the list. This specifies the entity that appears when the drop-down menu is expanded.

**Elements, Expanded Parent**

Drag an element onto the Expanded Parent box. This specifies the entity that serves as the parent when the drop-down menu is expanded. This is used to layer drop-down menus.

**Elements, Text Element**

Select an element from the list to specify the entity to display the text corresponding to the selected option.

**Elements, Icon Element**

Select an element from the list to specify the entity to display the icon corresponding to which option is selected.

**Options, Expand on Hover**

Select to enable drop-down behavior upon pause and collapse upon exit.
Options, Wait Time

Enter a number of seconds that the drop-down menu waits before expanding on pause or collapsing on exit.

Options, Collapse on Outside Click

Select to enable drop-down menu collapse upon clicking outside the menu.

Dropdown States, Expanded

Click plus to add states to the drop-down menu when it is expanded.

Actions, Expanded

Enter a text string to be sent as an action on the UI canvas when the drop-down menu is expanded.

Actions, Collapsed

Enter a text string to be sent as an action on the UI canvas when the drop-down menu is collapsed.

Actions, Option Selected

Enter a text string to be sent as an action on the UI canvas when a drop-down option is selected.

DropdownOption

You can use the DropdownOption component to make an element an option in a drop-down menu. When using the DropdownOption component, note the following:

- The DropdownOption is used along with the Dropdown component on an element.
- The DropdownOption requires an interactive component, typically a RadioButton component so that only one option can be selected at any time in the drop-down menu.
- Its child elements typically contain the Text component, which appears when the option is selected.

To see an in-game example of a completed canvas with the DropdownOption component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Interactable Components, Dropdown, Using UI Components and Dropdowns. Press Esc to exit the game.

To view this same canvas in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI \Canvases\LyShineExamples\Comp\Dropdown directory and open the UsingUi.uicanvas file. This canvas features a drop-down menu with a scroll box, image, check box, slider, and radio buttons.

To edit a DropdownOption component

In the Properties pane of the UI Editor (p. 2874), expand DropdownOption and do the following, as appropriate:

Interactable

See Properties (p. 2948) to edit the common interactive component settings.

Elements, Owning Dropdown

Select an element from the list that has the Dropdown component. This is the element to be modified when this option is selected.
Elements, Text Element

Select an element from the list that displays the text corresponding to this option. This text is displayed on the owning drop-down menu when this option is selected (as long as the drop-down menu has a Text Element configured).

Elements, Icon Element

Select an element from the list that displays the icon corresponding to this option. This icon is displayed on the owning drop-down menu when this option is selected (as long as the drop-down menu has an Icon Element configured).

Layout Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Like other UI components, layout components define the layout properties of your game's interface. The UI system features four layout components to organize your elements: LayoutColumn, LayoutRow, LayoutGrid, and LayoutCell. You can also nest layout components.

Topics

- LayoutColumn (p. 2973)
- LayoutRow (p. 2975)
- LayoutGrid (p. 2976)
- LayoutCell (p. 2978)
- LayoutFitter (p. 2982)
- Nesting Layout Components (p. 2983)

LayoutColumn

Add the LayoutColumn component to an element to make it a layout column. When you add child elements to the layout column, the layout column assigns each child element a layout cell. The layout column adjusts the size of the layout cells depending on how many child elements you add as well as the values provided by the child elements’ layout cells.

LayoutRow

Add the LayoutRow component to an element to make it a layout row. Like the layout column, the layout row assigns each child elements a layout cell. The layout row adjusts the size of the layout cells depending on how many child elements you add and the values provided by the child elements’ layout cells.

LayoutGrid

Add the LayoutGrid component to an element to make it a layout grid. The layout grid places child elements into a grid. Unlike the layout row and layout column, however, the layout grid does not use layout cells. The LayoutGrid component’s properties determine the size of its children.

LayoutCell

Add the LayoutCell component to a layout row or layout column's children to customize how a layout cell's size is determined. A layout cell is a programmatic concept whose properties define the area of a child element. Anytime that you add a child element to a layout row or layout column, that child element receives layout cell properties (not visible in the UI Editor), which determine the size
of the child's space. You can override the layout cell's calculated properties by adding the LayoutCell component to the child. For more information, see LayoutCell (p. 2978).

**LayoutFitter**

Add the LayoutFitter component to an element to make the element resize itself to fit its content. Use the layout fitter component with other components that provide cell sizing information, such as text, image (with ImageType set to Fixed), or layout components (cell, row, column, grid). For more information, see LayoutFitter (p. 2982).

**LayoutColumn**

You can use a LayoutColumn component to organize child elements into a column. To use this feature, add the LayoutColumn component to an element and then add child elements. The UI system positions the child elements within the column, from top to bottom or bottom to top, depending on the order you choose in the component properties. The child elements can contain a texture or image, a button, a check box, text, columns, rows, grids, and so on.

To see an in-game example of a completed canvas with the Layout Column component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Layout Components, Layout Column. You can view examples of different child sizes within a column. Press Esc to exit the game.

To view this same canvas in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI \Canvases\LyShineExamples\Comp\Layout directory and open the SimpleColumn.uicanvas file.

You can add a prebuilt Layout Column element from the slice library. When you do this, a simple layout column is automatically created and nested in your Hierarchy pane.

**To add a Layout Column element from the slice library**

- In the UI Editor (p. 2874), choose New, Element from Slice Library, LayoutColumn.

By default, the layout column gives every child the same amount of space, regardless of its content. You can, however, manipulate the sizes of each child by adding the layout cell (p. 2978) component to each or specific children.

The layout column can also give varying space to each child depending on its content. To enable the layout column to do this, clear the Ignore Default Cell option in the LayoutColumn component's properties.

In the first image, Ignore Default Cells is selected. The layout column gives each child the same amount of space regardless of their contents.
In the second image, **Ignore Default Cells** is cleared. The layout column calculates its children's space based on their contents.

To see an example of a completed canvas with the layout column component, open `SimpleColumn.uicanvas` in the LyShineExamples Gem (`\dev\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\`).

**To edit a layout column component**

In the **Properties** pane of the [UI Editor](p. 2874), expand **LayoutColumn** and do the following, as appropriate:
Padding

Type values in pixels, relative to the element's borders.

Spacing

Type values in pixels to adjust spacing between elements.

Order

Select Top-to-Bottom or Bottom-to-Top to specify the order in which the child elements appear in the column.

Child Alignment

If the layout's children don't occupy all the available layout space, use this setting to determine how the children are aligned.

For Horizontal, select Left, Center, or Right to determine how the children are aligned horizontally.

For Vertical, select Top, Center, or Bottom to determine how the children are aligned vertically.

Ignore Default Cell

Selected by default, this property causes the layout column to give each child an equal amount of space regardless of their contents (unless the child has a LayoutCell (p. 2978) component). The layout column ignores the layout cell's content-based default calculations.

When you clear this option, the layout column uses the children's layout cell calculated values to determine how much space to give each child based on its contents. For more information, see LayoutCell (p. 2978).

LayoutRow

You can use a LayoutRow component to organize child elements into a row. To use this feature, you add the LayoutRow component to an element and then add child elements. The UI system positions the child elements within the row, from left to right or right to left, depending on the order you choose. The child elements can contain a texture or image, a piece of text, a button, a check box, more columns, rows, grids, and so on. To control the sizes of specific, or all, children, add the layout cell (p. 2978) component to those children.

Similar to the LayoutColumn component, the LayoutRow component has an Ignore Default Cell property. For more information, see LayoutColumn (p. 2973).
To see an in-game example of a completed canvas with the **Layout Row** component, open the level **UiFeatures** in the project **SamplesProject**. Press **Ctrl+G** to play the game, and then choose **Components, Layout Components, Layout Row**. You can view examples of different child sizes within a row. Press **Esc** to exit the game.

To view this same canvas in the **UI Editor**, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout directory and open the **SimpleRow.uicanvas** file.

You can add a prebuilt **Layout Row** element from the slice library. When you do this, a simple layout row is automatically created and nested in your **Hierarchy** pane.

**To add a Layout Row element from the slice library**

- In the **UI Editor** (p. 2874), choose **New, Element from Slice Library, LayoutRow**.

**To edit a layout row component**

In the **Properties** pane of the **UI Editor** (p. 2874), expand **LayoutRow** and do the following, as appropriate:

- **Padding**
  
  Type values in pixels, relative to the element's borders.

- **Spacing**
  
  Type values in pixels to adjust spacing between elements.

- **Order**
  
  Select **Left-to-Right** or **Right-to-Left** to specify the order in which the child elements appear in the row.

- **Child Alignment**
  
  If the layout's children don't occupy all the available layout space, use this setting to determine how the children are aligned.

    - For **Horizontal**, select **Left**, **Center**, or **Right** to determine how the children are aligned horizontally.

    - For **Vertical**, select **Top**, **Center**, or **Bottom** to determine how the children are aligned vertically.

- **Ignore Default Cell**
  
  Selected by default, this property causes the layout row to give each child an equal amount of space regardless of their contents (unless the child has a **LayoutCell** (p. 2978) component). The layout row ignores the layout cell's content-based default calculations.

  When you clear this option, the layout row uses the children's layout cell calculated values to determine how much space to give each child based on its contents. For more information, see **LayoutCell** (p. 2978).

---

**LayoutGrid**

You can use a layout grid component to organize child elements into a uniform grid. To use this feature, you add the layout grid component to an element and then add child elements. The UI system positions the child elements in a grid pattern. You can choose whether the child elements are positioned left to right or right to left, and bottom to top or top to bottom. The child elements can contain a texture or image, a piece of text, a button, a check box, more columns, rows, grids, and so on. The size of each child is determined by the **Cell Size** property and is independent of each child's content.
To see an in-game example of a completed canvas with the LayoutGrid component, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Layout Components, Layout Grid. You can view examples of different fill patterns. Press Esc to exit the game.

To view this same canvas in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout directory and open the SimpleGrid.uicanvas file.

You can add a prebuilt Layout Grid element from the slice library. When you do this, a simple layout grid is automatically created and nested in your Hierarchy pane.

**To add a Layout Grid element from the slice library**

- In the UI Editor (p. 2874), choose New, Element from Slice Library, LayoutGrid.

**To edit a layout grid component**

In the Properties pane of the UI Editor (p. 2874), expand LayoutGrid and do the following, as appropriate:

**Padding**

Type values in pixels, relative to the element's borders.

**Spacing**

Type values in pixels to adjust spacing among elements.

**Cell size**

Type values in pixels to specify the size of the child elements.

**Order**

Do the following as appropriate:

- For Horizontal, select Left-to-Right or Right-to-Left to determine the order in which elements appear horizontally.
• For **Vertical**, select **Top-to-Bottom** or **Bottom-to-Top** to determine the order in which elements appear vertically.

• For **Starting With**, select **Horizontal** or **Vertical** to determine whether elements appear horizontally or vertically first.

**Child Alignment**

If the layout’s children don’t occupy all the available layout space, this setting determines how the children are aligned.

For **Horizontal**, select **Left**, **Center**, or **Right** to determine how the children are aligned horizontally.

For **Vertical**, select **Top**, **Center**, or **Bottom** to determine how the children are aligned vertically.

**LayoutCell**

When working with cell contents, it’s important to understand the difference between a layout cell and the **LayoutCell** component. A layout cell represents a set of values that determine the space or area allocated to a child in a layout row or column. The **LayoutCell** component, on the other hand, manipulates the way a layout cell is sized. A layout cell exists on a child of a layout row or layout column whether or not it has a **LayoutCell** component. The **LayoutCell** component simply provides a way to manipulate and override the default calculations of a layout cell.

A layout cell’s properties consist of a minimum size, a target size, and an extra size ratio. These properties are not directly modifiable in the UI Editor, but are determined in several ways:

• Components – The following components can affect the layout cell size:

  • Image or text – The image’s default size is the layout cell’s target size. The length and size of a string in a text component is the layout cell’s target size.

  • Layout row or layout column (added or nested as children) – The default values of a layout row or layout column, added as a child, determines the layout cell’s minimum and target size. The default value is calculated by the sum of its own children plus padding and spacing.

  **Note**
  
  The **LayoutColumn** and **LayoutRow** components contain a property called **Ignore Default Cells**. Selecting this property causes the above calculations to be ignored and simply allocate equal space to all children regardless of content. Clear this property to calculate layout cell values by components. For more information, see **LayoutColumn** (p. 2973).

• Fixed default layout cell values – If the child doesn’t have any components that calculate their own layout cell values, then the layout cell is assigned a minimum and target size of 0 and an extra size ratio of 1. This typically means equal spacing for the children that do not have a component affecting the layout cell’s size. Each layout cell grows at the same rate to fill the available space (hence the extra size ratio of 1).

• **LayoutCell** component – Add the **LayoutCell** component to specify values for the minimum and target sizes, and the extra size ratio. Any values you specify here override the values calculated by all other methods.

After layout cell values are calculated, layout cell space is allocated by the following:

1. First, each child receives its minimum size (**Min Height** or **Min Width**).
2. If space is available, each child receives its target size (**Target Height** or **Target Width**).
3. If space is still available after that, then the **Extra Size Ratio** value is used to determine how to allocate the remaining space. This ratio is relative to the child’s siblings. For example, if one child’s extra size ratio is 1, and another child’s is 2, then the second child gains twice as much extra space as
the first child. An extra size ratio of 0 means that no more space is allocated once the target size is reached.

Using the LayoutCell Component

You can apply the LayoutCell component to the children of a layout row or column to override the layout cell's default calculations.

In the following example, the layout column has three images as its children. The images each occupy equal space in the column.

If you add a LayoutCell component to the first image, and then select Min Height and assign a value of 100, then the UI system overrides that child's default calculated value, and gives the top image more height than its siblings, whose values are recalculated to adjust to the remaining column space.

In the next example, a layout grid was added as a child. Its calculated size is the same as its two siblings above it.
However, when you add a **LayoutCell** component to the grid, and then specify a **Min Height** of 100, then the grid, as a whole is granted that amount of space. If you add the **LayoutCell** component to the children of a layout grid, however, it has no effect. That's because individual grid spaces are always uniform and are controlled by the grid parent.
To edit a LayoutCell component

In the Properties pane of the UI Editor (p. 2874), expand LayoutCell and do the following, as appropriate:

**Min Width**

Select to define the layout cell's minimum width. Type a value in the box that appears.

**Min Height**

Select to define the layout cell's minimum height. Type a value in the box that appears.

**Target Width**

Select to define the layout cell's target width. Type a value in the box that appears. If space is available, this target width is allocated to the layout cell.

**Target Height**

Select to define the target height. Type a value in the box that appears. If space is available, this target height is allocated to the layout cell.
Extra Width Ratio

Select to define the layout cell's extra width ratio. Type a value in the box that appears. This value is a ratio that is relative to the other elements. If space remains after the layout cell reaches its target size, the Extra Size Ratio value is used to allocate the rest of the space.

Because this Extra Size Ratio value is relative to the other children, if one child's extra size ratio is 1, and another child's is 2, then the second child gains twice as much extra space as the first child. An extra size ratio of 0 means that no more space is allocated once the target size is reached.

Extra Height Ratio

Select to define the layout cell's extra height ratio. Type a value in the box that appears.

LayoutFitter

You can use the LayoutFitter component to make an element resize itself to fit its content. Use this component with other components that provide cell sizing information, such as the Text component, the Image component (with ImageType set to Fixed), or the Layout components (Cell, Row, Column, Grid).

To see an in-game example of a completed canvas with the LayoutFitter component, open the level UFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Layout Components, Layout Fitter. Press Esc to exit the game.

To view that same canvas in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI \Canvases\LyShineExamples\Comp\Layout directory and open the \fitter.uicanvas file.

To edit a layout fitter component

In the Properties pane of the UI Editor (p. 2874), expand LayoutFitter and do the following, as appropriate:
Horizontal Fit

Select the check box to resize an element's width to fit its content.

Vertical Fit

Select the check box to resize an element's height to fit its content.

Nesting Layout Components

You can nest layout components within other layout components.

To see an in-game example of a completed canvas with a nested layout, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Layout Components, Nested Layout. Press Esc to exit the game.

To view that same canvas in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout directory and open the \NestedLayout.uicanvas.

The following examples shows one large layout row.

Within the layout row are four columns.
Within column A, there are three layout rows. Column B has two layout grids. Column C has three images. Column D has one large image consisting of a color.

The first image in column C has a layout cell component with a minimum height set at 120. This gives it a larger space than its two siblings below it, which do not have LayoutCell components. Layout column D also has a LayoutCell component, with a minimum width of 110, giving it more space than the other three columns, which do not have LayoutCell components.
Tooltip Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add either a **Tooltip** component or a **TooltipDisplay** component to an element. With these components, you can display a tooltip when hovering over an interactive element.

To see in-game examples of completed canvases with Tooltip components, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Other Components, Tooltips. You can view examples of tooltip text options and display styles. Press Esc to exit the game.

To view these same canvases in the UI Editor, navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Tooltips directory.

You can open the following canvases:

- TextOptions.uicanvas
- Tooltips.uicanvas

**Tooltip**

You can use a **Tooltip** component to provide the text of the tooltip. Add a tooltip component to any interactive element that is to display a tooltip in the pause state.

**To edit a tooltip component**

1. In the Properties pane of the UI Editor (p. 2874), expand Tooltip.
2. Enter a text string.
TooltipDisplay

The **TooltipDisplay** component defines the tooltip's display behavior. Add a **TooltipDisplay** component to the element that is to visually represent the tooltip. You must also set the **Tooltip display element** (p. 2896) property of the canvas to this element. For more information, see **Configuring Canvas Properties** (p. 2894).

To edit a **TooltipDisplay** component

- In the **Properties** pane of the **UI Editor** (p. 2874), expand **TooltipDisplay** and use the following settings, as appropriate:

  **Trigger Mode (v1.24 and later)**

  Select a tooltip trigger condition:
  - **On Hover** – The tooltip appears when the pointer hovers over the interactive element, and disappears when the pointer moves off of the interactive element.
  - **On Press** – The tooltip appears when the interactive element is pressed and held, and disappears when the press is released. Note that the pointer might have moved elsewhere on the canvas by the time the release action has occurred.
  - **On Click** – The tooltip appears when a pointer click, which includes a press and a release, occurs on the interactive element. The tooltip disappears when the next pointer click occurs anywhere on the canvas. Note that if the pointer clicks on the same entity, the tooltip disappears, but then reappears after the specified **Delay time**.

  **Tip**
  On mobile devices, you might want to use **On Press** or **On Click** instead of **On Hover**.

  **Note**
  In all cases, the appearance of the tooltip is delayed by the amount of time specified in **Delay time**. Furthermore, in all cases, the tooltip will disappear after a fixed amount of time set by Lumberyard, regardless of other criteria specified in the trigger conditions.

  **Auto position**

  Automatically positions the element based on the positioning mode. The positioning mode is specified in the **Positioning** property.

  **Positioning**

  Select a positioning mode:
  - **Offset from mouse** – Position the element so that its pivot is a certain distance from the pointer. The distance is specified in the **Offset** property.
  - **Offset from element** – Position the element so that its pivot is a certain distance from the pivot of the element that triggered the tooltip display.

  **Offset**

  The offset to use when automatically positioning the element.

  **Auto size**

  Automatically resizes the element to match the tooltip string's size. The text element is a child of the element, and its text is specified in the **Text** property. If **Auto size** is selected, then the text element's anchors should be apart so that the text element can grow and shrink with its parent.

  **Text**

  The child element that is to display a tooltip string.
Dynamic Components

Delay time
The amount of time to wait before displaying the element.

Display time
The amount of time the element is to be displayed.

Show sequence
The animation sequence to be played when the element is about to appear.

Hide sequence
The animation sequence to be played when the element is about to disappear.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Dynamic components work together with Layout (p. 2972) components and the Scroll Box (p. 2962) component to display dynamic content in the user interface.

To see in-game examples of completed canvases with dynamic components, open the level UiFeatures in the project SamplesProject. Press Ctrl+G to play the game, and then choose Components, Dynamic Components. You can view examples of different types of dynamic layouts and scroll boxes. Press Esc to exit the game.

DynamicLayout

To use the DynamicLayout component, you place it on an element that also has a LayoutColumn (p. 2973), LayoutRow (p. 2975), or LayoutGrid (p. 2976) component. With the DynamicLayout component, you can change the number of children of the layout element at run time.

The layout element (1) dynamically resizes to fit its child elements. The first child (2) of the layout element acts as the prototype element. At run time, the UI system clones the prototype element to achieve the specified number of children in the layout.
The automatic resizing of the layout element depends on the layout type.

For **LayoutColumn** (p. 2973) and **LayoutRow** (p. 2975) elements, the layout element resizes in order to keep all of the child elements the same size as the prototype element.

For a **LayoutGrid** (p. 2976) element, the cell size of the **LayoutGrid** component determines the size of the child elements. The **LayoutGrid** element's initial size determines the number of children that can fit in each row or each column, depending on fill direction or **Order** settings. If the **Starting with** fill direction is **horizontal**, the UI system uses the **LayoutGrid** element's initial width to determine how many children fit in each row. If set to **vertical**, the initial height is used to determine how many children fit in each column.

![Layout Grid Component Settings](image)

**To use a dynamic layout component**

1. In the **UI Editor** (p. 2874), add a **LayoutRow**, **LayoutColumn**, or **LayoutGrid** prefab. To do this, choose **New, Element from Prefab**. Then select one of the layout elements.

   This serves as the structure or framework to hold your dynamic content.

2. Add a **DynamicLayout** component to your layout component. To do this, in the **Properties** pane choose **Add Component, DynamicLayout**.

   For **Num Cloned Elements**, enter the initial number of children to be created.

3. Create a child entity that has an **Image** component. To do this, right-click your layout component in the **Hierarchy** pane and choose **New, Element from Prefab, Image**.

   This image serves as the prototype element that will be cloned and filled with dynamic content.

**DynamicScrollBox**

To use the **DynamicScrollBox** component, you place it on an element that also has a **ScrollBox** component. With the **DynamicScrollBox** component, you can change the number of children of the scroll box's element at run time.

The content element dynamically resizes to fit its child elements. The first child of the content element acts as the prototype element. At run time, the UI system clones the prototype element to achieve the specified number of children in the layout.

With the **DynamicScrollBox** component, only the minimum number of child elements are actually created for display. This is different from the **DynamicLayout** component, where all child elements are created at run time and can consume a large amount of resources. The **DynamicScrollBox**'s elements are reused as the user scrolls; therefore, a scroll box can simulate a large number of children while maintaining good performance.
The **DynamicScrollBox** component automatically positions its children and resizes the content element to match the bounding box of its children. Each child's size is the same as the prototype element. By default, the children are positioned in a row from left to right. If vertical scrolling is enabled, the children are positioned in a column from top to bottom.

**To use a dynamic scroll box component**

1. In the **UI Editor (p. 2874)**, add a Scrollbox prefab. To do this, click **New, Element from Prefab, ScrollBox**. This serves as the structure or framework to hold your dynamic content.

2. Add a **DynamicScrollbox** component to your scroll box component. To do this, in the **Properties** pane choose **Add Component, DynamicScrollbox**. For **Default Num Elements**, enter the initial number of children to be created. This is mainly for previewing a canvas in **Preview** mode since the number of children ultimately comes from a custom component that implements the **UiDynamicScrollBoxDataBus**.

3. Create a child entity that has an **Image** component. To do this, right-click on your scroll box component in the **Hierarchy** pane, choose **New, Element from Prefab, Image**. This image serves as the prototype element that will be cloned and filled with dynamic content.

The **DynamicScrollBox** component uses a bus called **UiDynamicScrollBoxDataBus** to retrieve the number of children that the content element should have. It also uses a bus called **UiDynamicScrollBoxElementNotificationBus** to notify when a child element is about to become visible. This is where you set up the child element with dynamic content. To do this, you must create and add to the scroll box element a custom component that implements these two buses.

**EBus Interface**

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**UiDynamicScrollBoxDataBus:GetNumElements**

Implement this bus to provide a dynamic scroll box the number of children it should clone.

Returns the number of children that the dynamic scroll box should clone.

**Parameters**

None

**Return**

Number of children to clone.

**Ui DynamicScrollBoxElementNotificationBus:OnElementBecomingVisible**

Implement this bus to receive notifications when elements of a dynamic scroll box are about to become visible.

Sends a signal when an element of a dynamic scroll box is about to become visible.
Parameters

entityID – The entity ID of the element that is about to become visible.

index – The index of the element that is about to become visible.

Return

None

UISpawner Component

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the **UISpawner** component to spawn a runtime dynamic slice (**dynamicslice**) at an entity's location with an optional offset. In combination with scripting, you can use the **UISpawner** component to spawn any dynamic slice at any time and to spawn multiple instances of the same dynamic slice.

Contents

- Example UISpawner Component (p. 2990)
- UISpawner Component Properties (p. 2991)
- EBus Request Bus Interface (p. 2991)
  - Spawn (p. 2991)
  - SpawnRelative (p. 2991)
  - SpawnViewport (p. 2992)
  - SpawnSlice (p. 2992)
  - SpawnSliceRelative (p. 2992)
  - SpawnSliceViewport (p. 2992)
- EBus Notification Bus Interface (p. 2993)
  - OnSpawnBegin (p. 2993)
  - OnSpawnEnd (p. 2993)
  - OnEntitySpawned (p. 2993)
  - OnEntitiesSpawned (p. 2994)
  - OnTopLevelEntitiesSpawned (p. 2994)
  - OnSpawnFailed (p. 2994)

Example UISpawner Component

You can view a canvas with the **UISpawner** component in Samples Project. For more information, see Samples Project (p. 146).

To view an example canvas with a UISpawner component

1. For Samples Project, choose **File, Open, UI, UiFeatures**.
2. To start the game, press Ctrl+G.
3. Choose Components, Dynamic Components, Spawner. You can see an example of a simple button spawner and a more complex radio button spawner.
4. To exit the game, press Esc.

You can also view the example canvases in the UI Editor.

To view the canvases
1. In Lumberyard Editor, choose Tools, UI Editor.
2. Navigate to the lumberyard_version\dev\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Spawner directory.
3. Select a file and click Open.

UISpawner Component Properties

The UISpawner component has the following properties:

Dynamic Slice
- Select the slice asset to spawn.

Spawn on Activate
- If selected, spawns the selected slice upon activation.

EBus Request Bus Interface

Use the following request functions with the UISpawnerBus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

Spawn

Spawns the UI slice specified in the component at the entity’s location.

Parameters
- None

Return
- The slice instantiation ticket for this spawn.

Scriptable
- Yes

SpawnRelative

Spawns the UI slice specified in the component at the entity’s location with the specified relative offset.

Parameters
- Relative – The offset position from the entity with the spawner component.
**UISpawner Component**

**Return**

The slice instantiation ticket for this spawn.

**Scriptable**

Yes

**SpawnViewport**

Spawns the slice specified in the component at the specified viewport position.

**Parameters**

- **Pos** – The viewport position at which to spawn the slice.

**Return**

The slice instantiation ticket for this spawn.

**Scriptable**

Yes

**SpawnSlice**

Spawns the specified slice at the entity's location.

**Parameters**

- **slice** – Specifies the slice asset to be spawned.

**Return**

The slice instantiation ticket for this spawn.

**Scriptable**

No

**SpawnSliceRelative**

Spawns the given slice at the entity's location with the relative offset.

**Parameters**

- **slice** – Specifies the slice asset to be spawned.

  - **relative** – The offset position from the entity with the spawner component.

**Return**

The slice instantiation ticket for this spawn.

**Scriptable**

No

**SpawnSliceViewport**

Spawns the specified slice at the specified viewport position.
**Parameters**

- **slice** – Specifies the slice asset to be spawned.
- **pos** – The viewport position at which to spawn the slice.

**Return**

The slice instantiation ticket for this spawn.

**Scriptable**

Yes

**EBus Notification Bus Interface**

Use the following notification functions with the UiSpawnerNotificationBus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Working with the Event Bus (EBus) system (p. 1851).

**OnSpawnBegin**

Announces that the slice has been spawned, but entities have not yet been activated. **OnEntitySpawned** (p. 2993) events are about to be dispatched.

**Parameters**

- **ticket** – The slice instantiation ticket that is returned by the spawn function. These can be compared in order to know which spawn request it relates to.

**Scriptable**

Yes

**OnSpawnEnd**

Announces that a slice has been spawned. This function is called once for each spawn request. All **OnEntitySpawned** (p. 2993) events have been dispatched.

**Parameters**

- **ticket** – The slice instantiation ticket that is returned by the spawn function. These can be compared in order to know which spawn request it relates to.

**Scriptable**

Yes

**OnEntitySpawned**

Announces that an entity has been created during a spawn. This function is called once for each entity created while spawning a slice.

**Parameters**

- **ticket** – The slice instantiation ticket that is returned by the spawn function. These can be compared in order to know which spawn request it relates to.
- **spawnedEntity** – Specifies the ID of the spawned entity.
Other Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can add either or both of the fader and mask components to an element.

**Topics**
- Fader (p. 2995)
- Mask (p. 2995)
- Flipbook Animation (p. 2997)

**Fader**

You can use a Fader component to simultaneously adjust the transparency of an element and its children.

**To see in-game examples of completed canvases with a Fader component**

1. Open the Samples Project (p. 47).
2. When Lumberyard Editor has started, choose File, Open and open the UiFeatures level.
3. Press Ctrl+G to play the game.
4. Choose Components, Other Components, Fader. You can view an example of a direct fade and an animated fade.
5. Press Esc to exit the game.

**To view these same canvases in the UI Editor**

- Navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Fader directory.

You can open the following canvases:
- AnimFade.uicanvas
- DirectFade.uicanvas

**To edit a fader component**

1. In the Properties pane of the UI Editor (p. 2874), expand Fader.
2. For the Fade multiplier, use the slider to select a number between 0 (invisible) and 1 (opaque) and press Enter.

**Mask**

You can add a Mask component to an element to reveal only those parts of the child elements (for example, image or text) that are within the mask visual. In other words, the opaque region of the mask visual defines a shape that acts like a window through which you view the descendant elements.

**To see in-game examples of completed canvases with a Mask component**

1. Open the Samples Project (p. 47).
2. When Lumberyard Editor opens, expand the UI directory and open the level UiFeatures.
3. Press Ctrl+G to play the game.
4. Choose Components, Other Components, Mask.
5. Press Esc to exit the game.

To view these same canvases in the UI Editor

- Navigate to the `lumberyard_version/dev/Gems/LyShineExamples/Assets/UI/Canvases/LyShineExamples/Comp/Mask` directory.

You can open the following canvases:

- `AlphaMask.uicanvas` – Example of using a texture with alpha as a mask.
- `Basic.uicanvas` – Basic mask example.
- `ChildMaskElement.uicanvas` – Example of using a child element to draw the mask.
- `ImageMode.uicanvas` – Examples of using different image modes on the image used for the mask.
- `MaskingInteractables.uicanvas` – Example of how masks interact with components.
- `NestedMasks.uicanvas` – Example of nesting masks.
- `TextMask.uicanvas` – Example of using text as the visual component of the mask.

When you add a Mask component, the default mask is the visual component on that element, usually an Image component. If you want to use a nonrectangular mask, you must set this Image component to use a texture that contains an alpha channel (p. 3281), which specifies transparent and opaque areas. The child elements are masked by the mask visual. This means that the only parts of the child elements that are visible are the parts that are in the mask visual. In other words, the visible areas of the mask show the child elements, and the transparent areas of the mask visual hide the child elements. You can use other visual components, such as a Text component or Particle Emitter component to specify the mask visual.

You can make the mask movable to use in an animation. To do so without moving all of its children, you can use a special child element as a mask visual in addition to the visual component on the mask element. This child mask element can have as many children as you need to draw your mask visual.

Masks are commonly used with a ScrollBox prefab element (p. 2962).

To add simple mask with an Image component as the visual

1. In the UI Editor (p. 2874) toolbar, choose New, Empty Element. This is the parent element.
2. In the Properties pane, choose Add Component, Image to add an Image component.
3. Choose Add Component, Mask.
4. Right-click the parent element and then choose New, Empty Element to add a child element.
5. Select the child element and add to it an Image component.
6. In the Properties pane, click the folder icon next to Image, Sprite Path to select an image for the child element.
7. Open an image file in your current project directory.
8. On the parent element, in the Properties pane, click the folder icon next to the Image, Sprite Path to select the texture or image to use as a mask.
9. Open an image file in your current project directory. The image that you use as a mask should have opaque areas (which shows the content in child elements) and transparent areas (which hides the content in child elements).
10. In the Properties pane, under Mask, select Use alpha test.

To edit a Mask component

In the Properties pane of the UI Editor (p. 2874), expand Mask and specify the following parameters, as appropriate:
### Other Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Masking</td>
<td>If set, only the parts of the child elements that are revealed by the mask are visible. This parameter is enabled by default.</td>
</tr>
<tr>
<td>Draw behind</td>
<td>Draws the mask visual behind the child elements. This is useful for debugging.</td>
</tr>
<tr>
<td>Draw in front</td>
<td>Draws the mask visual in front of the child elements. This is useful for debugging.</td>
</tr>
<tr>
<td>Use alpha test</td>
<td>Uses the alpha channel in the masks visual's texture to define the mask. You must enable this parameter for masks that are anything other than a rectangle.</td>
</tr>
<tr>
<td>Mask interaction</td>
<td>Prevents input events from being sent to elements that are outside of the mask.</td>
</tr>
<tr>
<td>Child Mask Element</td>
<td>Specifies one of the child elements as a special element that is drawn as part of the mask visual. This parameter helps mask the remaining children.</td>
</tr>
</tbody>
</table>

### Flipbook Animation

You can use the `FlipbookAnimation` component to create simple image-based animations by animating the frames or cells of a sprite sheet (p. 2886).

**To see in-game examples of completed canvases with a FlipbookAnimation component**

1. Open the Samples Project (p. 47).  
2. When Lumberyard Editor has started, choose File, Open and open the UiFeatures level.  
3. Press Ctrl+G to play the game.  
4. Then choose Components, Other Components, Flipbook Animation.  
5. Choose None, Linear, or PingPong. Then click Start.  
6. Click Stop to stop the animation. Press Esc to exit.

**To view this canvas in the UI Editor**

1. Navigate to the \Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp \Flipbook directory.  
2. Open Flipbook.uicanvas.

You can add the `FlipbookAnimation` component to elements that also have an `Image` (p. 2936) component. You must also set that `Image` component to use a sprite file that has been configured as a sprite sheet (p. 2886).
To add and configure a FlipbookAnimation component

1. If you have not already done so, create a sprite sheet (p. 2886) for your animation image.
2. Add an **Image (p. 2936)** component.

   For the **SpriteType**, choose **Sprite/Texture asset**.

   In the **Sprite path**, click **Browse (…)** and navigate to the directory that contains the sprite sheet asset that you created. Select the sprite sheet.

3. Add a **FlipbookAnimation** component.

4. Configure the **FlipbookAnimation** properties:

   **Start Frame**
   
   The index of the sprite sheet cell that is to be the first frame in the animation. The value must be equal to or less than the **End Frame** value.

   **End Frame**
   
   The index of the sprite sheet cell that is to be the last frame in the animation. The value must be equal to or greater than the **Start Frame** value.

   **Loop start frame**
   
   The index of the sprite sheet cell that is to be the first frame in the looped portion of an animation. This value must be equal to or greater than **Start Frame** value and less than **End Frame** value. This setting has no effect if **Loop Type** is set to **None**.

   **Tip**
   
   To loop the entire animation, specify a value that is equal to the **Start Frame** value. To create an intro sequence that appears before the looping animation, specify a value greater than the **Start Frame**.

   **Loop Type**
   
   Includes the following options:
   
   - **None** – No looping behavior. The animation starts between the **Start Frame** and **End Frame** and then stops.
   - **Linear** – When the animation reaches **End Frame**, it loops by next playing the **Start Frame** and continues until the **End Frame**. This continues until the player stops it manually.
   - **PingPong** – Reverses the direction of the animation. After animation reaches the **End Frame** or the **Start Frame**, it reverses direction. The loop goes back and forth between the two frames until the player stops it.

   **Frame delay**
   
   Number of seconds to delay before displaying the next frame.

   **Auto Play**
   
   If enabled, automatically starts playing the flipbook animation when the canvas is loaded.

---

**Implementing New Fonts**

---

**Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.**

You can add fonts to your game UI in Lumberyard by saving the font asset to your game project and creating an .xml file that contains specifics for that font, such as the path to the font file and parameters that affect the font's appearance. You can combine multiple font assets into a single font family and further customize text appearance using **Text Markup (p. 2945)**.
Using the procedures in this section, you can:

- Add new fonts to your game UI (p. 3000)
- Create font families (p. 3000)
- Configure font rendering quality (p. 3002)

## Adding New Fonts

For each new font to add, you need the following files:

- A font asset – True Type Font (.ttf) or Open Type Font (.otf) file
- A font .font file describing the asset

### To add a new font to your UI

1. Save the font asset (.ttf or .otf file) to your game project directory, such as dev\SamplesProject\Font.
2. Copy an existing font .font file into your game project Font directory. The following directories (included in the Lumberyard project) contain font .font files for reference:
   - dev\Engine\Fonts\n   - dev\SamplesProject\Fonts\n3. Change the .font file name (leave the .font extension unchanged). Use any file name that is descriptive and appropriate for your purposes.
4. Open the .font file and edit the following line to point to your font asset file name.

```xml
<font path="yourFont.ttf" w="512" h="256"/>
```

After the Asset Processor has processed your font assets, you can select your font by loading the font .font file (p. 2943) in the **UI Editor** for any text component.

## Creating Font Families

You can combine multiple font assets into a single font family group.

The following is an example of a .fontfamily file.

```xml
<fontfamily name="MyFontFamily">
  <font>
    <file path="myfontfamily-regular.xml" />
    <file path="myfontfamily-bold.xml" tags="b" />
    <file path="myfontfamily-italic.xml" tags="i" />
    <file path="myfontfamily-bolditalic.xml" tags="b,i" />
  </font>
</fontfamily>
```

The UI system uses the font family definitions to determine which font asset to apply when styling text. You can combine the following types of assets:

- **Unstyled** – Font representing text with no styling applied. In the preceding example, this is myfontfamily-regular.xml.
- **Bold** – Font representing text with bold styling.
- **Italic** – Font representing text with italic styling.
• **Bold-Italic** – Font representing text with both bold and italic styling.

## Font Family File XML

To create a new font family file, you can create a new, empty plain text file and enter the contents, or you can modify an existing font family file.

### To add a new font family file to your UI

1. To create a new font family file, do one of the following:
   - Open Notepad (or similar program) and save an empty text file with a `.fontfamily` file extension.
   - Copy an existing `.fontfamily` file into your game project's Fonts directory.
2. Name your `.fontfamily` file appropriately (leave the `.fontfamily` extension).
3. Open your `.fontfamily` file and edit the contents to configure the font family.

   For example:

   ```xml
   <fontfamily name="MyFontFamily">
     <font>
       <file path="myfontfamily-regular.xml" />
       <file path="myfontfamily-bold.xml" tags="b" />
       <file path="myfontfamily-italic.xml" tags="i" />
       <file path="myfontfamily-bolditalic.xml" tags="b,i" />
     </font>
   </fontfamily>
   ```

   After the Asset Processor has processed your font assets, you can select your font family by selecting the *.fontfamily file in the UI Editor as the font for any text component. To apply custom styling to text using the font family, see Text Styling Markup (p. 2945).

The `.fontfamily` file uses XML. The UI system supports the following tags and attributes for the `.fontfamily` file:

**Tag:** `fontfamily`

**Attribute:** `name`

The unique name of the font family. Each font family name in a project must be unique, and only one `fontfamily` tag may be specified per `.fontfamily` file. You can, however, reuse the same font XML files (defined by the file tag) in multiple font families.

**Tag:** `font`

Container tag for the `file` tag.

**Attribute:** `lang`

The language that the font files should be associated with. The font files are loaded only if the listed language is being used. This enables a single font family to use different fonts and styling depending on the language being used.

**Tag:** `file`

**Attribute:** `path`

The path to the font XML, a TTF or OTF file. The path is relative to the font family file. The same font asset can be referenced multiple times for a given font family and across multiple font families.
**Attribute: tags**

This tag is optional. If omitted, this font file is used when no styling is applied.

Values:
- **b** – indicates `<b>` bold tag
- **i** – indicates `<i>` italic tag
- **b,i** – indicates when both `<b>` bold and `<i>` italic tags are applied

## Configuring Font Rendering Quality

Lumberyard's built-in UI system, **LyShine**, renders text using font textures. The quality of the on-screen text is affected by the font texture size, the number of character slots in the font texture, and the size of the text itself when rendered on the screen.

Use the procedures in this section to configure font size and texture to achieve quality text rendering.

### Font Texture Width and Height Attributes

Fonts are defined in XML by `.font` files. The XML in a `.font` file defines various parameters, such as the path to the source TTF/OTF asset and important rendering properties. The font file `Engine/Fonts/default-ui.font` included in the Lumberyard project has the following content.

```xml
<fontshader>
  <font path="Vera.ttf" w="512" h="256"/>
  <effect name="default">  
    <pass>
      </pass>
  </effect>
  <effect name="drop_shadow">  
    <pass>
      </pass>
    <pass>
      <color r="0" g="0" b="0" a="1"/>
      <pos x="1" y="1"/>
    </pass>
  </effect>
</fontshader>
```

The font texture resolution is controlled by the following line.

```xml
<font path="Vera.ttf" w="512" h="256"/>
```

In this example, the font texture has a resolution of 512x256. This resolution size (along with the number of character slots) is an important value for determining font rendering quality.

### Character Slots

In Lumberyard, a font texture is logically divided into equally sized slots. In each slot, there is a uniform amount of space for each character (glyph). By default (without additional configuration), there are 128 unique characters (16 rows * 8 columns).

If you support a language with many unique characters, such as Chinese, Japanese, or Korean, the default number of slots (128) might not be adequate for your needs, requiring further configuration. Otherwise, 128 unique characters might be adequate for most languages. The following information about character slots describes in further detail the font rendering pipeline in Lumberyard.
When rendering a string of characters, the number of unique characters in a string is different from the number of characters in a string (its length). The number of character slots in a font texture imposes a limitation only on the number of unique characters that can be rendered in a single frame.

For example, the following is a font definition which defines a font texture with 1 (1x1) texture slot.

```xml
<font path="Vera.ttf" w="512" h="512" widthslots="1" heightslots="1"/>
```

The default values for `widthslots` and `heightslots` is 16 and 8, respectively. However, as shown in the previous example, you can configure the number of character slots (1). This font can render a single unique character to the screen, any number of times, such as the following string.

AAAA

The number of unique characters in `AAAA` is 1, and the length of the string is 4. This font texture configuration can render this character an unlimited number of times (that is, a string of variable length) as long as the string contains only a single character. However, this font can't render the following string.

AABB

Because only one character slot exists in the texture, it can't store both the glyphs for upper-case 'A' and upper-case 'B' and render them both in a frame. To render this string, you need to increase the number of slots using the `widthslots` and `heightslots` parameters.

Here is another example.

```xml
<font path="NotoSansSC-Regular.otf" w="4096" h="4096" widthslots="128" heightslots="128"/>
```

In this example, the font texture size is 4096x4096, and there are a total number of 128x128 (16,384) character slots. To determine the available size for each character, divide the texture size (4096x4096) by the number of slots (128x128) to yield a 32x32 pixel space per character. This configuration enables you to render over 16,000 unique characters at a 32-pixel size in a single frame.

**Font Size**

Because a font texture is divided into a logical grid, a simple calculation determines how much real estate each character in the font can use:

- Font texture width / `widthslots` = slot width
- Font texture height / `heightslots` = slot height

Where `widthslots` is the number of character slots across the width (x-axis) of the font texture and `heightslots` is the number of character slots across the height (y-axis) of the font texture.

In the `default-ui.font` example in the previous section, the font texture size was 512x256. Assuming the character slots are at their default values (16x8):

- 512 / 16 = 32 (slot width)
- 256 / 8 = 32 (slot height)

For a 512x256 sized font texture, you can render pixel-perfect characters at 32x32 pixels.

Knowing these calculations helps you determine the right font texture size for the purposes of your game UI.

**To determine the right font texture size for your game UI**

1. Create the font `.font` file (p. 3000) for the font to use.
2. Choose an arbitrary font texture size to start with, such as 512x256 as used in the previous example.
3. Use the UI Editor to mock up a canvas with elements that have text components that use your font.
4. In the UI Editor’s Properties pane, under Text and then Font Size, experiment with the font size to find the ideal size for your use case.
5. After you have determined the appropriate font size for your purposes, use the following formula to determine the font texture width and height:
   - Texture width = Font size * widthslots
   - Texture height = Font size * heightslots

   **Note**
   The default value for widthslots and heightslots is **16** and **8**, respectively, which gives 128 total character slots. If you need to render more than 128 unique characters to the screen in a single frame—for example, if your game supports Chinese, Japanese, or Korean text—adjust these values accordingly.
6. Edit your font .font to use the calculated font texture size.

   **Note**
   - Font texture sizes don’t necessarily need to be a power of 2: 128, 256, 512, 1024, 2048, and so on. However, the width must be a multiple of widthslots (the default value is 16), and the height must be a multiple of heightslots (the default value is 8).
   - You can have multiple font .font files that reference the same TTF/OTF file but have different font texture sizes.

   For example, you might have some caption text that needs to appear only at a small font size, but you have other screens (perhaps a menu screen) where you want the same look and feel by using the same font. However, it needs to be larger and therefore needs a higher resolution font texture. You can achieve this with separate .font files for each use case, with font texture settings adjusted for ideal rendering quality.

### Configuring Font Properties

You can define the appearance of your UI font by configuring various properties that affect the font’s appearance and usage.

You define your font’s properties in the .font file, which is an XML file. The following example shows the default UI font XML file, located in Engine/Fonts/default-ui.font.
The default UI font XML uses the Vera font. It defines a font texture that can hold 128 unique character or glyphs that are 32x32 pixels. The font includes two representations that are defined with the `effect` tags `default` and `drop_shadow`. For the `default` effect, the font is rendered as is. For the `drop_shadow` effect, the font is first rendered as is. A second render pass produces the font in black with a 1-pixel offset from the first pass. This creates a basic shadowing effect of the characters.

**Note**

Fonts can be part of a font family (with a `.fontfamily` extension), though you can use a standalone font that isn't in a font family. For more information on font families, see [Creating Font Families](p. 3000).

Use the following tags, attributes, and values to define key features of your font.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>font</code></td>
<td>Contains critical attributes that define the path to the asset, size, and other font qualities.</td>
<td>See the section called “Font Tag Attributes” (p. 3006).</td>
</tr>
<tr>
<td><code>effect</code></td>
<td>Acts as a parent tag to <code>pass</code> children tags. Groups the <code>pass</code> tags that comprises the effect.</td>
<td><code>name</code> – Name of the effect.</td>
</tr>
<tr>
<td><code>effectfile</code></td>
<td>Specifies the path to an XML file that contains effect tags.</td>
<td><code>path</code> – String that specifies the path to an XML file containing effect tags.</td>
</tr>
<tr>
<td><code>pass</code></td>
<td>A child tag of an <code>effect</code> tag. You can add multiple <code>pass</code> tags as the children of a single <code>effect</code> tag.</td>
<td>Doesn't have attributes. Parents the following child tags that define text effects.</td>
</tr>
<tr>
<td></td>
<td>Defines a render pass of the text with various parameters that affect the text's rendering.</td>
<td>• <code>color</code> &lt;br&gt; • <code>pos</code> or <code>offset</code> &lt;br&gt; • <code>blend</code> or <code>blending</code></td>
</tr>
<tr>
<td></td>
<td>The <code>pass</code> tags can be layered on top of each other, giving the effect a unique look.</td>
<td></td>
</tr>
<tr>
<td><code>color</code></td>
<td>A text effect that is a child tag of the <code>pass</code> tag. Defines the text color.</td>
<td>Uses the following attributes to define the effect's intensity with float point values.</td>
</tr>
<tr>
<td></td>
<td>Minimum: 0.0 &lt;br&gt; Maximum: 1.0f &lt;br&gt; • <code>r</code> – Red. &lt;br&gt; • <code>g</code> – Green. &lt;br&gt; • <code>b</code> – Blue. &lt;br&gt; • <code>a</code> – Alpha. 0 is transparent. 1.0f is opaque.</td>
<td></td>
</tr>
<tr>
<td><code>pos</code> or <code>offset</code></td>
<td>A text effect that is a child tag of the <code>pass</code> tag. Sets the position of the text.</td>
<td>Uses the following attributes to set text position with integer values:</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Attributes</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>x</td>
<td>Offsets text along X axis relative to the position of the text</td>
<td>• x – Offsets text along X axis relative to the position of the text</td>
</tr>
<tr>
<td>y</td>
<td>Offsets text along Y axis relative to the position of the text</td>
<td>• y – Offsets text along Y axis relative to the position of the text</td>
</tr>
<tr>
<td>blend</td>
<td>A text effect that is a child tag of the pass tag. Defines alpha blending behavior of the text.</td>
<td>Uses the following attributes to define the text's alpha blending behavior:</td>
</tr>
<tr>
<td>blend or blending</td>
<td>A text effect that is a child tag of the pass tag. Defines alpha blending behavior of the text.</td>
<td>Uses the following attributes to define the text's alpha blending behavior:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• src – Source color value for text's alpha blending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• dst – Destination color value for text's alpha blending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• type – Alpha blending behavior with preconfigured settings determined by value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uses the following values for the src and dst attributes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• one</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• srcalpha or src_alpha</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• invsrcalpha or inv_src_alpha</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• dstalpha or dst_alpha</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• invdstalpha or inv_dst_alpha</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• dstcolor or dst_color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• srccolor or src_color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• invsrtcolor or inv_dst_color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• invsrtcolor or inv_src_color</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uses the following values for the type attribute:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modulate – src_alpha and inv_src_alpha (one minus source alpha) blending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• additive – src_alpha and one (for destination) blending</td>
</tr>
</tbody>
</table>

**Font Tag Attributes**

Font tag attributes define critical attributes for a font, such as the path to the TTF/OTF asset used to display the font, and other attributes affecting font render quality.
### Note
Most of these attributes have a direct impact on font rendering quality. For more information, see Configuring Font Rendering Quality (p. 3002).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
</table>
| path        | `<font path="Vera.ttf" ... />`             | Type: String  
Path to TTF or OTF font file asset.                                                                                                                        |
| fontsize    | `< ... fontsize="32"/>`                    | Type: Integer  
Defines in pixels the square size of the slots used to store glyphs (characters) in the font texture. For pixel-perfect render quality, this size should match the size specified when rendering the font. |
| w           | `< ... w="512" h="256" />`                 | Type: Integer  
Defines in pixels the width of the font texture.                                                                                                      |
| h           | `< ... w="512" h="256" />`                 | Type: Integer  
Defines in pixels the width of the font texture.                                                                                                      |
| widthslots  | `<font path="Vera.ttf" w="512" h="512" widthslots="8" heightslots="8" />` | Type: Integer  
Default: 16  
Defines the number of character or glyph slots along the X axis of the font texture.  
In the example, the font texture is 512x512. The width and height slots are set to 8. This gives a space of 64x64 for each character. |
| heightslots | `< ... widthslots="8" heightslots="8" />`  | Type: Integer  
Default: 8  
Defines the number of character or glyph slots along the y-axis of the font texture. |
| sizeratio   | `< ... sizeratio="0.6" />`                 | Type: Float  
Default: 0.8  
Applies uniform scaling to characters or glyphs when rendered into the font texture. |
### Configuring Font Properties

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sizebehavior</td>
<td><code>&lt;... sizebehavior=&quot;rerender&quot; /&gt;</code></td>
<td>The default scaling is usually ideal. You can adjust this value for fonts with unusual proportions, such as very long or wide fonts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value: rerender</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Renders text again at its new size. Improves font appearance quality when the text's rendered size differs from the font texture's glyph slot size. When text size changes, a simple transformation scale is applied. Quality degrades noticeably when text becomes larger or smaller. Rerendered text can improve that quality, depending on the font. Because rerendering takes time, it's not ideal for some cases, such as for animated text that changes sizes often.</td>
</tr>
<tr>
<td>hintbehavior</td>
<td><code>&lt;... hintbehavior=&quot;nohinting&quot; /&gt;</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configures hinting properties for the font.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* default – Hinting behavior as provided by the font. You can also omit the hintbehavior tag to use the default hinting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* nohinting – Disable hinting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* autohint – Hinting behavior procedurally derived from the font.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hintstyle</td>
<td><code>&lt; ... hintstyle=&quot;light&quot; /&gt;</code></td>
<td>Configures hinted text’s appearance. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• normal – Appearance as provided by the font. You can also omit the hintstyle tag to use the default hinting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• light – Fuzzier appearance that might more accurately represent the shape of the glyphs in the font.</td>
</tr>
<tr>
<td>smooth</td>
<td><code>&lt; ... smooth=&quot;blur&quot; smooth_amount=&quot;3&quot; /&gt;</code></td>
<td>Configures smoothing applied to the font. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• blur – Applies a simple blurring effect to characters or glyphs stored in the font texture. Use smooth_amount to set the number of iterations to use for blurring.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• supersample – Super samples characters or glyphs in the font texture by smooth_amount: 1 for 2x, and 2 for 4x.</td>
</tr>
<tr>
<td>smooth_amount</td>
<td><code>&lt; ... smooth=&quot;supersample&quot; smooth_amount=&quot;1&quot; /&gt;</code></td>
<td>Defines the amount of smoothing applied to the font. Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• none – Default. No smoothing applied.</td>
</tr>
</tbody>
</table>

**TrueType Fonts**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
CryFont is used to generate font textures that are required to render text on the screen. The various features of font rendering can be seen by using the \texttt{r\_DebugFontRendering} console variable. For more information, see Configuring Console Variables (p. 212).

The output is not only to test the functionality but also to document how the features can be used.

## Supported Features

CryFont supports the following features:

- **Font shaders** – Used to configure the appearance of fonts. Multiple passes with configurable offset and color are supported to enable generation of shadows or outlines. A sample font shader is shown in the following XML example.

  ```xml
  <fontshader>
  <font path="VeraMono.ttf" w="288" h="416"/>
  <effect name="default">
    <pass>
      <color r="0" g="0" b="0" a="1"/>
      <pos x="1" y="1"/>
    </pass>
  </effect>
  <effect name="console">
    <pass>
      <color r="0" g="0" b="0" a="0.5"/>
      <pos x="2" y="2"/>
    </pass>
  </effect>
  </fontshader>
  ``

  The attributes \texttt{w} and \texttt{h} of the XML font element specify the width and height of the font texture. The order of the passes in XML defines the order in which the passes are rendered. A \texttt{<pass>} element without child elements means that the pass is rendered with the default settings. The \texttt{<pos>} tag is used to offset the font, while the \texttt{<color>} tag is used to set font color and define the transparency (with the alpha channel \texttt{a}).

- Unicode – The default font used does not support all Unicode characters (to save memory), but other fonts can be used.

- TrueType fonts as source – Cached in a small texture. Common characters are pre-cached, but runtime updates are possible and supported.

- Colored text rendering

- Adjustable transparency

- Color variations within a string – Use a value of \texttt{0..9} to set one of the 10 available colors. Use \texttt{$$} to print the $ symbol, and \texttt{$o} to switch off the feature.

- Returns and tabs within a string

- Text alignment

- Computation of a string’s width and height – Used internally to handle center and right alignment.

- Font size variations – Bilinear filtering allows some blurring, but no mipmaps are used so this feature has limitations in minification.

- Proportional and monospace fonts

- Pixel-perfect rendering with exact texel-to-pixel mapping for best quality.
Useful Console Commands

The following console commands provide information about font rendering. For more information, see Configuring Console Variables (p. 212).

r_DebugFontRendering

Provides information on various font rendering features, useful for verifying function and documenting usage.
- 0=off
- 1=display

r_DumpFontNames

Logs a list of fonts currently loaded.

r_DumpFontTexture

Dumps the texture of a specified font to a bitmap file. You can use r_DumpFontTexture to get the loaded font names.

Animating the UI

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can animate your game's UI using one of the following tools:

Scripted Entity Tweener System (p. 3011)

Installed as a gem, this system uses custom Lua scripts to create powerful and flexible animations.

UI Animation Editor (p. 3023)

Embedded as a GUI in the UI Editor, this tool can create complex animations. You activate your animation sequences by using Script Canvas, with the UiAnimationBus eBus, or through C++ scripting.

Scripted Tweener Entity System

Lumberyard's Scripted Tweener Entity system is a script-driven animation system that creates powerful and flexible animations. To use this system, enable (p. 1064) the Scripted Tweener Entity (p. 1202) gem. In the Samples Project (p. 146), this gem is enabled by default.

The Scripted Tweener Entity system primarily animates entities in the UI Editor, and also supports the following Lumberyard component entities.
• Particle (p. 702)
• Lights – Point Light (p. 706), Area Light (p. 546), and Projector Light (p. 723).
• Mesh (p. 684)
• Transform (p. 878)

For a complete list, see Tweener Supported Components (p. 3018).

Topics
• Understanding Tweeners (p. 3012)
• Tweener Sample Level (p. 3017)
• Tweener Supported Components (p. 3018)
• Tweener Lua Script (p. 3020)
• Tweener Parameters (p. 3021)
• Tweener Timeline (p. 3022)

Understanding Tweeners

A tweener generates the transition between two images, giving the appearance of a smooth evolution between two frames. At its simplest, a tweener creates a linear transition. For example, a circle that starts at the top of the screen and ends at the bottom of the screen travels at a steady, unchanging pace between the two points. This is similar to an object moving on a conveyer belt. To simulate gravity, you can use a quadratic tweener, which accelerates slowly at first and then steeply increases towards the end. This simulates dropping an object that does not bounce.

Bounce and elastic tweeners offer flexibility when you want to simulate bounce or elasticity. The following animation shows a bouncing tweener. The ball starts offscreen and ends at the bottom of the screen. Between the start and end frames, the ball appears to bounce.
Properties other than location can also use tweens. The following animation shows a linear tweener for opacity, and a bouncing tweener for scale.
Tweeners are highly customizable. You can specify parameters to change properties such as duration, delay before starting, number of times to play, whether to go from the specified value to the current value or vice versa, and so on.

For available parameters and how to use them, see Tweener Supported Components (p. 3018). You can also create a timeline (p. 3022) to chain animations together with different tween types.

**Supported Tweener Easing Types**

The easiest way to visualize the concept of Scripted Entity Tweener easing types is to see it on a graph that depicts a change in position over time. The resulting line can be thought of as speed, though it isn't necessarily so.

The movement of objects in the natural world is rarely uniform. A rocket ship blasting off from the ground accelerates slowly as it first gains momentum, and then rapidly accelerates as it shoots toward the sky. If you viewed this motion on a position and time graph, it might look like a quartic or quintic easing in function. A bouncy ball dropped from a height accelerates toward the ground and hits it at a certain velocity. The ball bounces back up at that velocity and decelerates as it nears the top of its arc, and gravity pulls it back down for another bounce. Viewed on a position and time graph, looks like the bounce easing out function.

Lumberyard supports the following tweener ease methods (p. 3021) as easing in, easing out, and easing in-out (p. 3021) functions (ease type).

**Note**

The following images are referenced from http://easings.net.

**Linear**

Does not ease in or out. No acceleration or deceleration. Constant movement, such as a conveyor belt in motion.
**Quadratic**

Easing in function starts at a zero velocity, then accelerates.

Easing out function begins at a certain velocity and decelerates to zero.

Easing in-out function starts at zero velocity, accelerates until the halfway point, and then decelerates to zero.

The following graphs show quadratic ease in, quadratic ease out, and quadratic ease in-out functions.

---

**Cubic**

Similar to quadratic, but with a steeper curve, which indicates a slower rate of acceleration or deceleration at first, followed by a rapid acceleration or deceleration.

The following graphs show cubic ease in, cubic ease out, and cubic ease in-out functions.

---

**Quart**

Similar to cubic, but with an even slower rate of acceleration or deceleration followed by an even faster rate of acceleration or deceleration.

The following graphs show quartic ease in, quartic ease out, and quartic ease in-out functions.

---

**Quint**

Similar to quart, but with an even slower rate of acceleration or deceleration followed by an even faster rate of acceleration or deceleration.

The following graphs show quintic ease in, quintic ease out, and quintic ease in-out functions.

---

**Sine**

Based on a sine or cosine function. Gentle easing in or easing out, with an almost constant speed like the linear function.

The following graphs show sinusoidal ease in, sinusoidal ease out, and sinusoidal ease in-out functions.
Exponential

Similar to cubic.

The following graphs show exponential ease in, exponential ease out, and exponential ease in-out functions.

![Graphs of exponential ease in, ease out, and ease in-out functions.]

Circular

Based on the equation for half of a circle. The graph in the following image is stretched vertically so that you can see how the graph looks like a part of a circle.

The following graphs show circular ease in, circular ease out, and circular ease in-out functions.

![Graphs of circular ease in, ease out, and ease in-out functions.]

Elastic

An example of an elastic easting out function is a plucked guitar string. The string moves up and down with decreasing frequency until it comes to a rest. Easing in is the same motion but in reverse.

The following graphs show elastic ease in, elastic ease out, and elastic ease in-out functions.

![Graphs of elastic ease in, ease out, and ease in-out functions.]

Back

An example of a back easing in function is a toy car pulled backward to wind the springs, and then released.

The following graphs show back ease in, back ease out, and back ease in-out functions.

![Graphs of back ease in, ease out, and ease in-out functions.]

Bounce

Depicts a bouncing motion. For example, a rubber ball dropped from a height displays a bounce ease out function as it bounces and eventually comes to rest.

The following graphs show bounce ease in, bounce ease out, and bounce ease in-out functions.

![Graphs of bounce ease in, ease out, and ease in-out functions.]

Version 1.28
3016
Tweener Sample Level

You can use the Samples Project (p. 146) to see an example level that uses tweener.

To see example Scripted Entity Tweeners in action

1. In Lumberyard Editor, in the Samples Project (p. 146), open the level Samples \ScriptedEntityTweenerSample\SampleFullscreenAnimation.

2. To switch to gameplay mode, press Ctrl+G. The header, list items, earth, stars, and spaceship use tweener for their animation.

3. Press Esc to exit.
To view the sample UI canvas
1. Open the UI Editor (p. 2874).
2. Choose File, Open Canvas.
3. Navigate to the `lumberyard_version\dev\SamplesProject\UI\Canvases\ScriptedEntityTweenerSample` directory and then select `MenuAnimateInSample.uicanvas`.

To view the sample Lua Script with the tweener code
1. In the Hierarchy pane, select MenuSample.
2. In the Properties pane, under the Lua Script component, for Script, click the {} icon. This opens the samplescreenanimation script in the Lua Editor.

3. See how the script uses the listed parameters. For a complete list of parameters, see Tweener Parameters (p. 3021).

Try the following:
- Modify the value of the easeMethod (p. 3021) parameter, especially on the blocks that include `id = self.Properties.CharacterImage`, which represent the spaceship.
- Increase or decrease the duration parameter.
- Change the starting position ["x"] and ["y"].
4. Save your changes and close the Lua Editor.
5. In the UI Editor, click Preview in the upper right to see how your changes affect the game UI.

Tweener Supported Components

You can use the Scripted Entity Tweener system for any entity parameter currently recognized by the tweener system.

You can see the list of supported entity parameters in the `lumberyard_version\dev\Gems\ScriptedEntityTweener\Assets\Scripts\ScriptedEntityTweener\ScriptedEntityTweener.lua` file in the table `self.animationParameterShortcuts`.

The following example shows the `self.animationParameterShortcuts` table that is inside of the `ScriptedEntityTweener.lua` file.

```lua
self.animationParameterShortcuts =
{
  --UI Related
  ["opacity"] = {"UiFaderComponent", "Fade"},
  ["imgColor"] = {"UiImageComponent", "Color"},
```

Version 1.28
3018
["layoutMinWidth"] = {"UiLayoutCellComponent", "MinWidth" },
["layoutMinHeight"] = {"UiLayoutCellComponent", "MinHeight" },
["layoutTargetWidth"] = {"UiLayoutCellComponent", "TargetWidth" },
["layoutTargetHeight"] = {"UiLayoutCellComponent", "TargetHeight" },
["layoutExtraWidthRatio"] = {"UiLayoutCellComponent", "ExtraWidthRatio" },
["layoutExtraHeightRatio"] = {"UiLayoutCellComponent", "ExtraHeightRatio" },
["layoutColumnPadding"] = {"UiLayoutColumnComponent", "Padding" },
["layoutColumnSpacing"] = {"UiLayoutColumnComponent", "Spacing" },
["layoutRowPadding"] = {"UiLayoutRowComponent", "Padding" },
["layoutRowSpacing"] = {"UiLayoutRowComponent", "Spacing" },
["scrollHandleSize"] = {"UiScrollBarComponent", "HandleSize" },
["scrollHandleMinPixelSize"] = {"UiScrollBarComponent", "MinHandlePixelSize" },
["scrollValue"] = {"UiScrollBarComponent", "Value" },
["sliderValue"] = {"UiSliderComponent", "Value" },
["sliderMinValue"] = {"UiSliderComponent", "MinValue" },
["sliderMaxValue"] = {"UiSliderComponent", "MaxValue" },
["sliderStepValue"] = {"UiSliderComponent", "StepValue" },
["textSize"] = {"UiTextComponent", "FontSize" },
["textCharacterSpace"] = {"UiTextComponent", "CharacterSpacing" },
["textSpacing"] = {"UiTextComponent", "LineSpacing" },
["textInputSelectionColor"] = {"UiTextInputComponent", "TextSelectionColor" },
["textInputCursorColor"] = {"UiTextInputComponent", "TextCursorColor" },
["tooltipDelayTime"] = {"UiTooltipDisplayComponent", "DelayTime" },
["tooltipDisplayTime"] = {"UiTooltipDisplayComponent", "DisplayTime" },
["pivotX"] = {"UiTransform2dComponent", "PivotX" },
["pivotY"] = {"UiTransform2dComponent", "PivotY" },
["x"] = {"UiTransform2dComponent", "LocalPositionX" },
["y"] = {"UiTransform2dComponent", "LocalPositionY" },
["rotation"] = {"UiTransform2dComponent", "Rotation" },
["w"] = {"UiTransform2dComponent", "LocalWidth" },
["h"] = {"UiTransform2dComponent", "LocalHeight" },

-- 3d transform
["3position"] = {"TransformComponent", "Position" },
["3rotation"] = {"TransformComponent", "Rotation" },
["3dscale"] = {"TransformComponent", "Scale" },
-- Camera
["camFov"] = {"CameraComponent", "FieldOfView" },
["camNear"] = {"CameraComponent", "NearClipDistance" },
["camFar"] = {"CameraComponent", "FarClipDistance" },

--[[
-- Some available virtual properties without shortcuts
-- Lights
[""] = {"LightComponent", "Visible" },
["] = {"LightComponent", "Color" },
["] = {"LightComponent", "DiffuseMultiplier" },
["] = {"LightComponent", "SpecularMultiplier" },
["] = {"LightComponent", "Ambient" },
["] = {"LightComponent", "PointMaxDistance" },
["] = {"LightComponent", "PointAttenuationBulbSize" },
["] = {"LightComponent", "AreaMaxDistance" },
["] = {"LightComponent", "AreaWidth" },
["] = {"LightComponent", "AreaHeight" },
["] = {"LightComponent", "AreaFOV" },
["] = {"LightComponent", "ProjectorMaxDistance" },
["] = {"LightComponent", "ProjectorAttenuationBulbSize" },
["] = {"LightComponent", "ProjectorFOV" },
["] = {"LightComponent", "ProjectorNearPlane" },
["] = {"LightComponent", "ProbeAreaDimensions" },
["] = {"LightComponent", "ProbeSortPriority" },
["] = {"LightComponent", "ProbeBoxProjected" },
Tweener Lua Script

You must have a minimum set of code to play an animation in the Scripted Entity Tweener system. As described in Tweener Sample Level (p. 3017), you add this code to a Lua script. You can copy and paste the code into a text file, and then change the file name extension to .lua. You use this script as part of a Lua script component. For more information about Lua script components, see Adding Lua Scripts to Component Entities (p. 2684).

In the following example, the entity's opacity is linearly tweened to 0.5 over 5 seconds.

```lua
local AnimateUiEntity =
{
    Properties =
    {
        ToAnimate = {default = EntityId()},
    },
}

function AnimateUiEntity:OnActivate()
    self.tickBusHandler = TickBus.Connect(self)
    self.ScriptedEntityTweener =
        require("Scripts.ScriptedEntityTweener.ScriptedEntityTweener")
end

function AnimateUiEntity:OnTick(deltaTime, timePoint)
    self.tickBusHandler:Disconnect()
    self.ScriptedEntityTweener:StartAnimation
    {
        id = self.Properties.ToAnimate,
        duration = 5.0,
        ["opacity"] = 0.5,
    }
end

function AnimateUiEntity:OnDeactivate()
    self.ScriptedEntityTweener:OnDeactivate()
end

return AnimateUiEntity
```

The following example shows a full call with all the default parameters. In this example, the entity moves to the ["x"] and ["y"] positions indicated in the code over the duration of 5 seconds.

```lua
self.ScriptedEntityTweener:StartAnimation
{
    id = self.Properties.ToAnimate,
    duration = 5.0,
```
Scripted Tweener Entity System

```
{["x"] = 150, ["y"] = 200,
 timeIntoTween = 0, -- Start tween some seconds in
 easeMethod = ScriptedEntityTweenerEasingMethod_Linear,
 easeType = ScriptedEntityTweenerEasingType_In,
 delay = 0,
 timesToPlay = 1,
 isFrom = false,
 isPlayingBackward = false,
 uuid = Uuid.Create(),
 --onComplete = function() Debug.Log("Called when this animation is done") end
 --onUpdate = function(currentValue, currentProgressPercent) Debug.Log("Called when this
 animation updates") end
 --onLoop = function() Debug.Log("Looped animation") end
}
```

For the parameter descriptions, see Tweener Supported Components (p. 3018).

**Tweener Parameters**

Scripted Entity Tweener parameters provide flexibility and allow you to customize your tweener animations. Use the parameters with a timeline (p. 3022) for a variety of possibilities.

Use the following tweener parameters to customize your animation.

**duration**

Specifies the time in seconds for the animation to go from the start value to its final value.

**["x"] and ["y"]**

Shortcuts for the x and y values of a UiTransform2dComponent, which is automatically attached to every UI entity. For a full list of shortcuts, see `lumberyard_version\dev \Gems\ScriptedEntityTweener\Assets\Scripts\ScriptedEntityTweener \ScriptedEntityTweener.lua`.

**timeIntoTween**

Specifies a tween to begin at a specified point (in seconds). For example, if the duration is set to 6, and timeIntoTween is set to 3, then the tween begins immediately at its halfway point and finishes in three more seconds.

**easeMethod**

Specifies the easing type (p. 3014) to apply to the tween.

- ScriptedEntityTweenerEasingMethod_Linear
- ScriptedEntityTweenerEasingMethod_Quad
- ScriptedEntityTweenerEasingMethod_Cubic
- ScriptedEntityTweenerEasingMethod_Quart
- ScriptedEntityTweenerEasingMethod_Quint
- ScriptedEntityTweenerEasingMethod_Sine
- ScriptedEntityTweenerEasingMethod_Expo
- ScriptedEntityTweenerEasingMethod_Circ
- ScriptedEntityTweenerEasingMethod_Elastic
- ScriptedEntityTweenerEasingMethod_Back
- ScriptedEntityTweenerEasingMethod_Bounce

**easeType**

Modifies easeMethod.
- `ScriptedEntityTweenerEasingType_In`
- `ScriptedEntityTweenerEasingType_Out`
- `ScriptedEntityTweenerEasingType_InOut`

**delay**

Specifies an amount of time in seconds to delay the start of the animation.

**timesToPlay**

Number of times to play the animation. Specify -1 to play the animation indefinitely.

**isFrom**

If false, the animation starts at the value specified in the component and ends at the value specified in the script.

If true, the animation starts at the value specified in the script and ends at the value specified in the component.

**isPlayingBackward**

If true, the animation plays backwards. To play an animation backward, you must specify a value for `timeIntoTween`. If a value is not specified for `timeIntoTween`, this parameter is ignored.

**uuid**

ID generated by the Scripted Entity Tweener system and used for debugging purposes. You do not need to modify or specify a value.

**onComplete, onUpdate, and onLoop**

Use these parameters to specify a callback function when the animation completes, updates, or loops. `onUpdate` also passes back the animation’s current completion state percentage and the current value of the animated parameter.

**Tweener Timeline**

In the Scripted Entity Tweener system, you can create a timeline for granular control over your animations. You can chain and group animations. You can pause, resume, seek, play backward, label, and even dynamically control the playback speed of the animations.

To create a timeline, you can customize the following Lua script.

In this example, the script first animates the entity specified by `ToAnimate`, increasing its size from its original value to a width `"w"` and height `"h"` of 600. The script then animates the entity by shrinking it to a width `"w"` and height `"h"` of 200.

```lua
local AnimateUiEntity = 
{ 
    Properties = 
    { 
        ToAnimate = {default = EntityId()}, 
    },
}

function AnimateUiEntity:OnActivate() 
    self.tickBusHandler = TickBus.Connect(self) 
    self.ScriptedEntityTweener = 
        require("Scripts.ScriptedEntityTweener.ScriptedEntityTweener") 
end
```
function AnimateUiEntity:OnTick(deltaTime, timePoint)
    self.tickBusHandler:Disconnect()

    self.timeline = self.ScriptedEntityTweener:TimelineCreate()

    self.timeline:Add(
        {
            id = self.Properties.ToAnimate,
            easeMethod = ScriptedEntityTweenerEasingMethod_Bounce,
            easeType = ScriptedEntityTweenerEasingType_In,
            duration = 3.0,
            ["w"] = 600, ["h"] = 600,
        }
    )

    self.timeline:Add(
        {
            id = self.Properties.ToAnimate,
            easeMethod = ScriptedEntityTweenerEasingMethod_Sine,
            easeType = ScriptedEntityTweenerEasingType_in-out,
            duration = 3.0,
            ["w"] = 200, ["h"] = 200,
        }
    )

    self.timeline:Play()
end

function AnimateUiEntity:OnDeactivate()
    self.ScriptedEntityTweener:OnDeactivate()
end

return AnimateUiEntity

The following example shows all of the supported timeline operations.

-- Chained animations
self.timeline = self.ScriptedEntityTweener:TimelineCreate()
self.timeline:Add(...)  
self.timeline:Add(...)  
self.timeline:Play()

--Labels
self.timeline:AddLabel("Label", 0.5) -- Add label that specifies 0.5 seconds into animation
self.timeline:Play("Label") -- Play from label

--General timeline operations
local animationParameters = { id = self.Properties.ToAnimate, duration = 3, ["opacity"] = 0.5 }
self.timeline:Add(animationParameters), {["label"] = "LabelName"}, {["offset"] = 2} -- Start animation 2 seconds after LabelName
self.timeline:Add(animationParameters), {["initialStartTime"] = 2}, {["offset"] = -1} -- Start animation at second 1

self.timeline:Play()
self.timeline:Play(2) -- Play timeline starting at 2 seconds
self.timeline:Pause() -- Pause timeline
self.timeline:Resume() -- Resumes timeline, Play() also resumes
self.timeline:Seek(3) -- Play timeline starting at 3 seconds
self.timeline:PlayBackwards() -- Start playing animation backwards
self.timeline:PlayBackwards(3) -- Start playing animation backwards starting from second 3

UI Animation Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use animation sequences to animate UI elements in your game's UI. A UI canvas can contain many named animation sequences.

The UI Animation Editor has the following features:

- **Menu** – Operations for creating new animation sequences and switching between the Track Editor and Curve Editor.
- **Toolbar** – Tools for the editing and playback of animations. The Curve Editor displays an additional toolbar at the top of the pane.
- **Node pane** – Area for showing the active sequence and all of the elements that it is animating. A track for each animated property appears underneath the related element.
- **Editor pane** – Area for either the Track Editor, the Curve Editor, or both.

**To show the Animation Editor if it is not already visible**

- From the UI Editor (p. 2874) menu, choose View, Animation Editor.

To create an animation sequence, you first create a new sequence, assign one or more UI elements to it, and then record changes you make to the UI element(s)—this becomes the animation sequence. You can then edit the animation sequence(s) using the Animation Editor. These processes are described in greater detail in the following sections.

**Recording Animation Data**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Recording animation typically involves three steps:

1. Create a new animation sequence.
2. Add a UI element to that sequence.
3. Turn on animation recording to capture changes in the element properties.

Adding a UI element also adds a node to the sequence. After that any time that you enter record mode, a track is automatically added to your animation for any change you make to this UI element. You do not need to manually add tracks. For more information, see Using the Node Pane (p. 3026).

You can create an animation sequence from the Animation Editor menu or toolbar.

To create a new animation sequence

In the Animation Editor (p. 3023), do one of the following:

• From the Sequence menu, choose New Sequence.
• Click the Add Sequence icon on the toolbar.

To add a UI element to the sequence

1. In the UI Editor (p. 2874), select the UI element that you want to animate.
2. In the Animation Editor, right-click the sequence that you created and click Add Selected UI Element(s).

To record an animation sequence

1. In the Animation Editor toolbar, click the Record icon.
2. In the UI Editor, use either the Properties pane or viewport pane to make changes to the selected UI element.
3. After making all changes, click the Stop icon in the Animation Editor toolbar.

Note

In the current release, not all component properties can be recorded. For example, enumerated values, such as the image type of an image component, cannot be animated.

After you record a track, it appears beneath its UI element. The node pane lists your current animation sequences. For more information on the Node Pane, see Using the Node Pane (p. 3026)

Playing Animation Sequences

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can play back the animation in the Animation Editor to preview what it will look like in your game. Playing the animation sequence animates the UI elements in the UI Editor.

To control playback of animation in the UI Editor

• In the Play toolbar of the Animation Editor, use the Play, Pause, Stop, Go to start of sequence, and Go to end of sequence buttons.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you create your sequence(s) and record animation data to them, you can use the Node Pane, Track Editor, and Curve Editor in the Animation Editor to modify your sequences.

- In the Node Pane, you can add or remove UI elements from an animation sequence, edit sequences, and work with keys. For more information, see Using the Node Pane (p. 3026).
- In the Track Editor, you can limit your animation preview, manipulate keys, and change your animation's timeline. For more information, see Using the Track Editor (p. 3028).
- In the Curve Editor, you can manipulate splines to change the behavior of the transitions between keys. For more information, see Using the Curve Editor in the UI Animation Editor. (p. 3031).

Topics
- Using the Node Pane (p. 3026)
- Using the Track Editor (p. 3028)
- Using the Curve Editor in the UI Animation Editor. (p. 3031)

Using the Node Pane

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Node Pane in the Animation Editor (p. 3023) displays all the nodes in the selected animation sequence. Each item listed in the Node Pane is considered a node, though they represent different parts of the sequence. You can use the Node Pane to add or delete UI element nodes. Track nodes appear beneath its UI element when you record a track.

The animation sequence node, at the top level, contains a list of its UI elements nodes. Each UI element node contains a list of its track nodes.

1. Animation Sequence node
2. **UI Element** nodes

3. **Track** nodes

To add a new UI element node

1. In the **UI Editor** (p. 2874), select one or more elements.
2. In the **Animation Editor**, right-click anywhere in the node pane and select **Add Selected UI Element(s)**.

To remove a UI element node

- In the **Animation Editor**, in the node pane, right-click an element node and click **Delete**.

To edit a track

1. In the **Animation Editor**, in the node pane, select a track node.
2. Right-click the track node and choose any of the following:
   - Copy Keys
   - Copy Selected Keys
   - Paste Keys
   - Disable the track

You can also use the **Edit Sequence** tool to edit the properties of the sequence directly. You can set various properties, such as the start and end time, whether the sequence loops, and so on.

To open the **Edit Sequence** tool

- In the **Animation Editor**, click the **Edit Sequence** icon.
Using the Track Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Track Editor displays all the tracks in your current animation sequence. The Track Editor enables you to do the following:

- Move, delete, copy, and paste keys
- Change the timeline of the animation
- Control the animation preview range
To display the Track Editor

- In the Animation Editor (p. 3023), choose View, Track Editor or View, Both.

To zoom in or out

- Scroll the mouse wheel

To pan the view

- With the mouse in the Track Editor, drag using the middle mouse button

Topics

- Working with Keys in the Track Editor (p. 3029)
- Moving the Play or Record Point in the Track Editor (p. 3030)
- Previewing in the Track Editor (p. 3030)

Working with Keys in the Track Editor

When you create an animation, key values are automatically recorded. Using the Track Editor, you can move, delete, copy, and paste keys. Keys are represented by a green circle on the timeline of each track.

To move a key

- Click a key and drag it to a new time on the timeline.

To constrain movement to time only

- Hold Shift as you drag the key to a new time on the timeline.

To scale the selected keyframes while moving a key

- Hold Alt as you drag the key to a new time on the timeline.

To delete a key

- Right-click a key and click Delete.

To copy a key

- Right-click a key and click Copy.

To paste a key

- Right-click in the timeline and click Paste. Move the key to the desired point on the timeline, then click to place.

The Track Editor's toolbar features a variety of tools to improve your workflow efficiency when editing tracks. Pause over each icon to reveal the tooltips.

Some of the toolbar functions require you to select multiple keys.
To select multiple keys

- In the Track Editor, drag to select multiple keys. The selected keys appear as white circles.

You can also use the Track Editor toolbar to select, move, and snap keys. When moving keys, you can choose to snap them to other keys, to frames, or to second ticks.

Working with Keys in the Track Editor Toolbar

<table>
<thead>
<tr>
<th>Toolbar icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to previous key</td>
<td>Selects the key directly before the currently selected key.</td>
</tr>
<tr>
<td>Go to next key</td>
<td>Selects the key directly after the currently selected key.</td>
</tr>
<tr>
<td>Slide keys</td>
<td>Moves the currently selected key and slides all the keys after it to the new point on the timeline while maintaining the original spacing.</td>
</tr>
<tr>
<td>Move keys</td>
<td>Moves the currently selected key(s) to the new point on the timeline without affecting other keys.</td>
</tr>
<tr>
<td>Scale keys</td>
<td>Functions only with multiple keys selected to increase or decrease the space between the selected keys proportionally.</td>
</tr>
<tr>
<td>Magnet Snapping</td>
<td>Snaps to keys in other tracks as you get close to them; allows you to place the key anywhere but indicates a red circle on the key you want to snap to.</td>
</tr>
<tr>
<td>Frame Snapping</td>
<td>Snaps to frames.</td>
</tr>
<tr>
<td>Tick Snapping</td>
<td>Snaps to second ticks.</td>
</tr>
</tbody>
</table>

Moving the Play or Record Point in the Track Editor

The play or record point of the animation sequence is shown as a vertical magenta slider on the timeline. Move the play or record point, and the properties of the UI elements in the Animation Editor (p. 3023) change to the values specified by the animation tracks.

To move the play or record point in the Track Editor

- Click or drag the vertical magenta slider in the timeline.

Previewing in the Track Editor

The Track Editor features a timeline along its top edge. To preview your entire animation, simply click the Play button to play your animation at its normal speed. You can also change the speed of preview by clicking the arrow beside the play button and selecting 2, 1, ½, ¼, or ⅛. You can also limit your animation preview, as it plays, to a specific time frame.

To limit play preview in the Track Editor

1. In the timeline, at the start of your preferred preview time, right-click to mark the time with a red triangle.
2. In the timeline, at the end of your preferred preview time, right-click again to mark the end time with a red triangle.
3. Click the Play button to preview your animation in the time frame specified.
**Note**
When you preview an animation or move the playback position on the timeline, it moves the UI elements in the **UI Editor**. This means that, if you then save the canvas, these UI elements will be saved in this position.
Reposition the timeline or preview a different sequence to position the UI elements at the positions in which you want them to load before you save the canvas.

**Using the Curve Editor in the UI Animation Editor.**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Curve Editor** displays animations as function curves. Each track's curves represent an animation of a property value (such as anchor, offset, color, or any property of a UI element).

**The elements of a curve**
1. Curve or spline
2. Spline key
3. Tangent handles

The path of the curve represents the transition of the value between the keyframes. If the value changes in a straight line between each keyframe (linear), transitions between keyframes will not be smooth. The default curve causes the value to smoothly ease in and ease out. Each key has an in tangent and an out tangent. Depending on the preferred effect, you can use the toolbar icons to switch the tangents to auto, zero, step, or linear. You can also manually drag the tangent handles.
By default, animation tracks are recorded with a smooth transition. You can use the buttons in the toolbar at the top of the Curve Editor to change how the curves behave on either side of the selected key. You can also drag spline keys to a different point in the timeline.

**To display the Curve Editor**

- In the UI Animation editor, choose View, Curve Editor or View, Both.

**To zoom in or out**

- Scroll the mouse wheel

**To pan the view**

- With the mouse in the Curve Editor, drag using the middle mouse button

**To adjust a spline key**

1. In the Node Pane, select a track. The curves for that track appear in the Curve Editor.
2. In the Curve Editor, select a spline key.
3. Do one or more of the following:
   - Drag the spline key to a different point on the timeline.
   - Use the toolbar buttons to select a preset: auto, zero, step, or linear.

You can select multiple spline keys to modify at once. Once selected, you can move them all together, set their in and out tangents, and so on.

**To select multiple spline keys**

- In the Curve Editor, drag a selection box over all the spline keys you want to select.

**To edit multiple elements at once**

1. In the Animation Editor's node pane, select the track that contains the subtracks that you want to edit.
2. Drag a selection box over the spline keys that you want to select.
3. Drag spline keys to edit their position.

---

**UI Lua Reference**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use the following Lua scripting functions when you work with UI canvases in Lumberyard Editor. For non-UI component entity Lua scripting functions, see the Component Entity Lua API Reference (p. 2705).

**Topics**
LyShineLua.ShowMouseCursor

- Controls the visibility of the mouse cursor. 1 specifies that the mouse cursor is displayed; 0 specifies that it is hidden.
UIButtonComponent

Syntax

void ShowMouseCursor(bool visible)

Services messages for the UIButtonComponent.

GetOnClickActionName

Returns the action triggered when the button is released.

Syntax

AZStd::string UIButtonBus::GetOnClickActionName()

SetOnClickActionName

Sets the action triggered when the button is released.

Syntax

void UIButtonBus::SetOnClickActionName(const AZStd::string& actionName)

UIButtonNotificationBus

Services notifications for the UIButtonComponent.

OnButtonClick

Occurs when a button has been clicked.

Syntax

void OnButtonClick()

UICanvasComponent

Contains buses and notification buses for UI canvas and animation.
UiCanvasBus

Services messages for the UiCanvasComponent.

FindObjectByName

Returns the entity ID of the first UI element in this canvas that has the specified name.

Syntax

```
AZ::EntityId FindElementByName(const AZStd::string& name)
```

ForceEnterInputEventOnInteractable

Forces a specified interactive element to receive an Enter press/release navigation event. This is useful for automated testing using script, such as simulating a button click.

Syntax

```
void ForceEnterInputEventOnInteractable(AZ::EntityId entityId)
```

ForceHoverInteractable

Forces a specified interactive element to receive the hover.

Syntax

```
void ForceHoverInteractable(AZ::EntityId entityId)
```

GetChildElement

Returns the entity ID of the child element that has the specified index.

Syntax

```
AZ::EntityId GetChildElement(int index)
```

GetChildElements

Returns a list of entity IDs of the child elements of the canvas.

Syntax

```
AZStd::vector<AZ::EntityId> GetChildElements()
```

GetDrawOrder

Returns the draw order on the canvas.

Syntax

```
int GetDrawOrder()
```
GetEnabled
Returns the enabled flag of the canvas. True if the canvas is enabled; false otherwise. Enabled canvases are updated and rendered for each frame.

Syntax
```cpp
bool GetEnabled()
```

GetHoverInteractable
Returns the entity ID of the interactive element that has the hover.

Syntax
```cpp
AZ::EntityId GetHoverInteractable()
```

GetIsNavigationSupported
Returns true if the canvas accepts navigation input; false otherwise.

Syntax
```cpp
bool GetIsNavigationSupported()
```

GetIsPixelAligned
Returns true if pixel alignment is enabled for this canvas; false otherwise. If pixel alignment is enabled, then all corners of all elements are rounded to the nearest pixel when they are rendered.

Syntax
```cpp
bool GetIsPixelAligned()
```

GetIsRenderToTexture
Returns true if the canvas is rendered to a texture instead of the full screen; false otherwise.

Syntax
```cpp
bool GetIsRenderToTexture()
```

GetKeepLoadedOnLevelUnload
Returns true if the canvas stays loaded when the level is unloaded; false otherwise.

Syntax
```cpp
bool GetKeepLoadedOnLevelUnload()
```

GetNumChildElements
Returns the number of child elements that the canvas has.

Syntax
int GetNumChildElements()

**GetRenderTargetName**

Returns the name of the texture that is created when this canvas is rendered to a texture.

**Syntax**

```c++
AZStd::string GetRenderTargetName()
```

**GetTooltipDisplayElement**

Returns the entity ID of the tooltip element that is to be displayed when the pointer pauses on an interactable element.

**Syntax**

```c++
AZ::EntityId GetTooltipDisplayElement()
```

**RecomputeChangedLayouts**

Tells the layout manager for the canvas to recompute the layout. This happens automatically every frame. Occasionally a script modifies something that affects the layout (such as the reparenting of an element) and tries to query positions in the same frame. In this case you can call `RecomputeChangedLayouts` to force an immediate recalculation of all layouts on the canvas that have been flagged for recomputing.

**Syntax**

```c++
void RecomputeChangedLayouts()
```

**SetDrawOrder**

Sets the draw order on the canvas.

**Syntax**

```c++
void SetDrawOrder(int drawOrder)
```

**SetEnabled**

Sets whether the canvas is enabled. Enabled canvases are updated and rendered for each frame. Specify true to enable the canvas, false to disable.

**Syntax**

```c++
void SetEnabled(bool enabled)
```

**SetHoverInteractable**

Sets the interactable element on the canvas on which the pointer has been paused.

**Note**

This action is done automatically, but in unusual situations it can be useful to call this function from a script.
### Syntax

```cpp
void SetHoverInteractable(AZ::EntityId entityId)
```

**SetIsNavigationSupported**

Specify true to make the canvas accept navigation input; false otherwise.

**Syntax**

```cpp
void setIsNavigationSupported(bool isSupported)
```

**SetIsPixelAligned**

Specify true to enable pixel alignment for this canvas; false otherwise.

**Syntax**

```cpp
void setIsPixelAligned(bool isPixelAligned)
```

**SetIsRenderToTexture**

Sets whether the canvas is rendered to a texture instead of the full screen. Specify true to render the canvas to a texture; false otherwise.

**Syntax**

```cpp
void setIsRenderToTexture(bool isRenderToTexture)
```

**SetKeepLoadedOnLevelUnload**

Sets whether the canvas stays loaded when the level is unloaded. Specify true to keep the canvas loaded on level unload; false otherwise.

**Syntax**

```cpp
void setKeepLoadedOnLevelUnload(bool keepLoaded)
```

**SetRenderTargetName**

Sets the name of the texture that is created when this canvas is rendered to a texture.

**Syntax**

```cpp
void setRenderTargetName(const AZStd::string& name)
```

**SetTooltipDisplayElement**

Sets the element that is to be displayed when the pointer pauses over an interactable element.

**Syntax**

```cpp
void setTooltipDisplayElement(AZ::EntityId entityId)
```
UiCanvasInputNotifications

Services notifications for the UiCanvasComponent.

OnCanvasEnterPressed

Called when the “enter” key is pressed. Sends the entity that was pressed or an invalid entity id if no element was pressed.

Syntax

```cpp
OnCanvasEnterPressed(AZ::EntityId entityId)
```

OnCanvasEnterReleased

Called when the enter key is released. Sends the entity that was pressed or an invalid entity id if no element was pressed.

Syntax

```cpp
OnCanvasEnterReleased(AZ::EntityId entityId)
```

OnCanvasHoverEnd

Called when an element ends being hovered.

Syntax

```cpp
OnCanvasHoverEnd(AZ::EntityId entityId)
```

OnCanvasHoverStart

Called when an element starts being hovered.

Syntax

```cpp
OnCanvasHoverStart(AZ::EntityId entityId)
```

OnCanvasPrimaryPressed

Called on a positional input press. Sends the entity that was pressed or an invalid entity id if no element was pressed.

Syntax

```cpp
void OnCanvasPrimaryPressed(AZ::EntityId entityId)
```

OnCanvasPrimaryReleased

Called on a positional input release. Sends the entity that was pressed or an invalid entity id if no element was pressed.

Syntax
OnCanvasPrimaryReleased(AZ::EntityId entityId)

UiCanvasNotificationBus
Services notifications for the UiCanvasComponent.

OnAction
Maps the name of an action to the entity ID of a canvas component that triggers the action.

Syntax

void OnAction(AZ::EntityId entityId, const AZStd::string& actionName)

UiAnimationBus
Animation messages serviced by the UiCanvasComponent.

AbortSequence
Aborts the specified sequence.

Syntax

void AbortSequence(const AZStd::string& sequenceName)

GetSequencePlayingSpeed
Returns the current playing speed of the specified sequence.

Syntax

float GetSequencePlayingSpeed(const AZStd::string& sequenceName)

GetSequencePlayingTime
Returns the current playing time of the specified sequence.

Syntax

float GetSequencePlayingTime(const AZStd::string& sequenceName)

IsSequencePlaying
Returns true if the specified sequence is playing; false otherwise.

Syntax

bool IsSequencePlaying(const AZStd::string& sequenceName)

PauseSequence
Pauses the specified sequence.
**Syntax**

```c
void PauseSequence (const AZStd::string& sequenceName)
```

**ResetSequence**

Resets the specified sequence to the start.

**Syntax**

```c
void ResetSequence (const AZStd::string& sequenceName)
```

**ResumeSequence**

Causes the specified sequence to resume.

**Syntax**

```c
void ResumeSequence (const AZStd::string& sequenceName)
```

**SetSequencePlayingSpeed**

Sets the current playing speed of the specified sequence.

**Syntax**

```c
void SetSequencePlayingSpeed(const AZStd::string& sequenceName, float speed)
```

**StartSequence**

Starts playing the specified sequence.

**Syntax**

```c
void StartSequence(const AZStd::string& sequenceName)
```

**StopSequence**

Stops playing the specified sequence.

**Syntax**

```c
void StopSequence (const AZStd::string& sequenceName)
```

**UiAnimationNotificationBus**

Services animation notifications for the UICanvasComponent.

**OnUiAnimationEvent**

Specifies the animation event for the specified sequence.

**Syntax**
void OnUiAnimationEvent(eUiAnimationEvent uiAnimationEvent, AZStd::string animSequenceName)

Following are possible values for the eUiAnimationEvent flag.

enum eUiAnimationEvent
{
    eUiAnimationEvent_Started,
    eUiAnimationEvent_Stopped,
    eUiAnimationEvent_Aborted,
    eUiAnimationEvent_Updated
};

UICanvasManager

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Contains functions for loading, unloading, and finding UI canvases.

UICanvasManagerBus

Services messages for the UiCanvasManager component.

FindLoadedCanvasByPathName

Returns the entity ID of the loaded canvas that has the specified canvas pathname.

Syntax

AZ::EntityId FindLoadedCanvasByPathName(const AZStd::string& canvasPathname)

LoadCanvas

 Loads a canvas and returns the entity ID of the loaded canvas.

Syntax

AZ::EntityId LoadCanvas(const AZStd::string& canvasPathname)

UnloadCanvas

 Unloads a UI canvas.

Syntax

void UnloadCanvas(AZ::EntityId canvasEntityId)

UICheckboxComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
You can use this component to implement a UI check box in Amazon Lumberyard.

**UiCheckboxBus**

Services messages for the UiCheckboxComponent.

**GetChangedActionName**

Returns the action triggered when the check box value changes.

**Syntax**

```cpp
AZStd::string & GetChangedActionName()
```

**GetCheckedEntity**

Returns the child element that is shown when the check box is selected.

**Syntax**

```cpp
AZ::EntityId GetCheckedEntity()
```

**GetState**

Returns the state of the check box. True if selected; false otherwise.

**Syntax**

```cpp
bool GetState()
```

**GetTurnOffActionName**

Returns the action triggered when the check box is cleared.

**Syntax**

```cpp
const AZStd::string & GetTurnOffActionName()
```

**GetTurnOnActionName**

Returns the name of the action that is triggered when the check box is selected.

**Syntax**

```cpp
const AZStd::string& GetTurnOnActionName()
```

**GetUncheckedEntity**

Returns the child element that is shown when the check box is cleared.

**Syntax**

```cpp
AZ::EntityId GetUncheckedEntity()
```
**SetChangedActionName**
Sets the action triggered when the check box value changes.

**Syntax**

```cpp
void SetChangedActionName(const AZStd::string& actionName)
```

**SetCheckedEntity**
Sets the child element to show when the check box is selected.

**Syntax**

```cpp
void SetCheckedEntity(AZ::EntityId entityId)
```

**SetState**
Sets the state of the check box. True if selected; false otherwise.

**Syntax**

```cpp
void SetState(bool checked)
```

**SetTurnOffActionName**
Sets the action triggered when the check box is cleared.

**Syntax**

```cpp
void SetTurnOffActionName(const AZStd::string & actionName)
```

**SetTurnOnActionName**
Sets the action triggered when the check box is selected.

**Syntax**

```cpp
void SetTurnOnActionName(const AZStd::string & actionName)
```

**SetUncheckedEntity**
Sets the child element to show when the check box is cleared.

**Syntax**

```cpp
void SetUncheckedEntity(AZ::EntityId entityId)
```

**ToggleState**
Toggles the state of the check box.

**Syntax**
**UiCursorBus**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The `UiCursorBus` contains functions for manipulating the cursor in the game UI.

**IncrementVisibleCounter**

Increments the UI cursor visible counter. Should be paired with a call to `DecrementVisibleCounter`.

**Syntax**

```c
void IncrementVisibleCounter()
```

**DecrementVisibleCounter**

Decrements the UI cursor visible counter. Should be paired with a call to `IncrementVisibleCounter`.

**Syntax**

```c
void DecrementVisibleCounter()
```

**IsUiCursorVisible**

Queries whether the UI cursor is currently visible.

**Syntax**

```c
bool IsUiCursorVisible()
```

**SetUiCursor**

Sets the UI cursor image.

**Syntax**

```c
```
void SetUiCursor(const char* cursorImagePath)

**GetUiCursorPosition**

Returns the UI cursor position (in pixels) relative to the top left corner of the UI overlay viewport.

**Syntax**

```
AZ::Vector2 GetUiCursorPosition()
```

**UIDraggableComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use a draggable component to make an element draggable for drag-and-drop behavior.

**UiDraggableBus**

Services messages for the UiDraggableComponent.

**GetCanDropOnAnyCanvas**

Gets the flag that indicates whether this draggable can be dropped on this canvas or any loaded canvas.

**Syntax**

```
bool GetCanDropOnAnyCanvas()
```

**GetDragState**

Returns the current drag state of the draggable component.

**Syntax**

```
eUiDragState GetDragState()
```

Following are possible values for eUiDragState.

```
enum eUiDragState
{
    eUiDragState_Normal,
    eUiDragState_Valid,
    eUiDragState_Invalid
};
```

**GetOriginalFromProxy**

Gets the original draggable component from a draggable component that is acting as a proxy.

**Syntax**
**IsProxy**

Checks whether a draggable component is acting as a proxy for another draggable component.

**Syntax**

```cpp
bool IsProxy()
```

**ProxyDragEnd**

Concludes the drag of a proxy. Call `ProxyDragEnd` at the end of a drag if `SetAsProxy` was used for the drag. Call `ProxyDragEnd` from the `OnDragEnd` handler of the proxy element. This results in a call to `OnDragEnd` for the original draggable component.

**Syntax**

```cpp
void ProxyDragEnd(AZ::Vector2 point)
```

**RedoDrag**

Causes the draggable component to redetect the drop targets that are underneath the pointer and resend `OnDropHoverStart` or `OnDropHoverEnd` messages if needed. You can call `RedoDrag` from a script after the script has caused drop targets to change positions. This function is most useful for keyboard or gamepad navigation.

**Syntax**

```cpp
void RedoDrag(AZ::Vector2 point)
```

**SetAsProxy**

Sets a draggable component to act as a proxy for another draggable component and starts dragging the draggable component at the specified point. A proxy is useful if you want the visible element being dragged to be on a canvas other than the one on which the original draggable component is selected. It is also useful if you want to avoid moving the original draggable component.

**Syntax**

```cpp
void SetAsProxy(AZ::EntityId originalDraggableId, AZ::Vector2 point)
```

**SetCanDropOnAnyCanvas**

Sets the flag that indicates whether a draggable can be dropped on this canvas or any loaded canvas.

**Syntax**

```cpp
void SetCanDropOnAnyCanvas(bool anyCanvas)
```

**SetDragState**

Sets the current drag state of the draggable component.
Syntax

```csharp
void SetDragState(eUiDragState dragState)
```

For possible values for `eUiDragState`, see `GetDragState (p. 3046).

**UiDraggableNotificationBus**

Services notifications for the `UiDraggableComponent`.

**OnDrag**

Occurs each time the drag position changes during dragging. `OnDrag` events happen only between `OnDragStart` and `OnDragEnd` events.

**Syntax**

```csharp
void OnDrag(AZ::Vector2 position)
```

**OnDragEnd**

Occurs at the end of dragging when the release input event occurs. The `OnDragEnd` notification is sent before the `OnDrop` drop target notification.

**Syntax**

```csharp
void OnDragEnd(AZ::Vector2 position)
```

**OnDragStart**

Occurs when dragging is detected on the draggable component. For mouse or touch input, this occurs when movement has been detected after the press or touch.

**Syntax**

```csharp
void OnDragStart(AZ::Vector2 position)
```

**UiDropdownComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use `UiDropdownComponent` to implement a UI dropdown menu in Amazon Lumberyard.

**UiDropdownBus**

Services messages for the `UiDropdownComponent`.

**GetValue**

Returns the value of the dropdown. The value is the last option that was selected.

**Syntax**
UiDropdownComponent

**GetValue**

Sets the value of the dropdown manually.

**Syntax**

```cpp
AZ::EntityId GetValue()
```

**SetValue**

Returns the content element that this dropdown expands.

**Syntax**

```cpp
AZ::EntityId GetContent()
```

**GetContent**

Sets the content element that this dropdown expands.

**Syntax**

```cpp
void SetContent(AZ::EntityId content)
```

**GetExpandOnHover**

Returns whether this dropdown should expand automatically on hover.

**Syntax**

```cpp
bool GetExpandOnHover()
```

**SetExpandOnHover**

Sets whether this dropdown expands automatically on hover.

**Syntax**

```cpp
void SetExpandOnHover(bool expandOnHover)
```

**GetWaitTime**

Returns how long to wait before expanding upon hover and collapsing upon exit.

**Syntax**

```cpp
float GetWaitTime()
```

**SetWaitTime**

Sets how long to wait before expanding upon hover and collapsing upon exit.
**Syntax**

```cpp
void SetWaitTime(float waitTime)
```

**GetCollapseOnOutsideClick**

Returns whether this dropdown collapses when the user clicks outside.

**Syntax**

```cpp
bool GetCollapseOnOutsideClick()
```

**SetCollapseOnOutsideClick**

Sets whether this dropdown collapses when the user clicks outside.

**Syntax**

```cpp
void SetCollapseOnOutsideClick(bool collapseOnOutsideClick)
```

**GetExpandedParentId**

Returns the element that the dropdown content parents to when expanded (the root element by default).

**Syntax**

```cpp
AZ::EntityId GetExpandedParentId()
```

**SetExpandedParentId**

Sets the element that the dropdown content parents to when expanded.

**Syntax**

```cpp
void SetExpandedParentId(AZ::EntityId expandedParentId)
```

**GetTextElement**

Returns the text element that displays the text of the currently selected option.

**Syntax**

```cpp
bool GetCollapseOnOutsideClick()
```

**SetTextElement**

Sets the text element that displays the text of the currently selected option.

**Syntax**

```cpp
voidSetTextElement(AZ::EntityId textElement)
```
GetIconElement

Returns the icon element that displays the icon of the currently selected option.

**Syntax**

```cpp
AZ::EntityId GetIconElement()
```

SetIconElement

Sets the icon element that displays the icon of the currently selected option.

**Syntax**

```cpp
void SetIconElement(AZ::EntityId iconElement)
```

Expand

Expands the dropdown menu.

**Syntax**

```cpp
void Expand()
```

Collapse

Collapses the dropdown menu.

**Syntax**

```cpp
void Collapse()
```

GetExpandedActionName

Returns the name of the action that is sent when the dropdown is expanded.

**Syntax**

```cpp
const LyShine::ActionName& GetExpandedActionName()
```

SetExpandedActionName

Sets the name of the action that is sent when the dropdown is expanded.

**Syntax**

```cpp
void SetExpandedActionName(const LyShine::ActionName& actionName)
```

GetCollapsedActionName

Returns the name of the action that is sent when the dropdown is collapsed.

**Syntax**

```cpp
```
const LyShine::ActionName& GetCollapsedActionName()

SetCollapsedActionName
Sets the name of the action that is sent when the dropdown is collapsed.

Syntax

void SetCollapsedActionName(const LyShine::ActionName& actionName)

GetOptionSelectedActionName
Returns the name of the action that is sent when the dropdown value is changed.

Syntax

const LyShine::ActionName& GetOptionSelectedActionName()

SetOptionSelectedActionName
Sets the name of the action that is sent when the dropdown value is changed.

Syntax

void SetOptionSelectedActionName(const LyShine::ActionName& actionName)

UiDropdownNotificationBus
Services notifications for the UiDropdownComponent.

OnDropdownExpanded
Notifies that the dropdown was expanded.

Syntax

void OnDropdownExpanded()

OnDropdownCollapsed
Notifies that the dropdown was collapsed.

Syntax

void OnDropdownCollapsed()

OnDropdownValueChanged
Notifies that an option was selected.

Syntax

void OnDropdownValueChanged(AZ::EntityId option)
UiDropdownOptionComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use `UiDropdownOptionComponent` to implement a dropdown options for a UI element in Amazon Lumberyard.

UiDropdownOptionBus

Services messages for the `UiDropdownOptionComponent`.

GetOwningDropdown

Returns the owning dropdown to be modified when this dropdown option is selected.

Syntax

```plaintext
AZ::EntityId GetOwningDropdown()
```

SetOwningDropdown

Sets the owning dropdown to be modified when this dropdown option is selected.

Syntax

```plaintext
void SetOwningDropdown(AZ::EntityId owningDropdown)
```

GetTextElement

Returns the text element that is used to display this dropdown option’s text.

Syntax

```plaintext
AZ::EntityId GetTextElement()
```

SetTextElement

Sets the text element that is used to display this dropdown option’s text.

Syntax

```plaintext
void SetTextElement(AZ::EntityId textElement)
```

GetIconElement

Returns the icon element that is used to display this dropdown option’s icon.

Syntax

```plaintext
AZ::EntityId GetIconElement()
```
**SetIconElement**

Sets the icon element that is used to display this dropdown option's icon.

**Syntax**

```cpp
void SetIconElement(AZ::EntityId iconElement)
```

**UiDropdownOptionNotificationBus**

Services notifications for the `UiDropdownOptionComponent`.

**OnDropdownOptionSelected**

Notifies that this dropdown option was selected.

**Syntax**

```cpp
void OnDropdownOptionSelected()
```

**UiDropTargetComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use a drop target component to make an element a target for drag-and-drop behavior.

**UiDropTargetBus**

Services messages for the `UiDropTargetComponent`.

**GetDropState**

Returns the current drop state of the drop target component.

**Syntax**

```cpp
eUiDropState GetDropState()
```

Following are possible values for `eUiDropState`.

```cpp
enum eUiDropState
{
    eUiDropState_Normal,
    eUiDropState_Valid,
    eUiDropState_Invalid
};
```

**GetOnDropAction**

Returns the action triggered when a draggable component is dropped on this drop target.
Syntax

```cpp
const AZStd::string& GetOnDropAction()
```

**SetDropState**

Sets the current drop state of the drop target component.

**Syntax**

```cpp
Void SetDropState(eUiDropState dragState)
```

For possible values for `eUiDropState`, see GetDropState (p. 3054).

**SetOnDropAction**

Sets the action triggered when a draggable component is dropped on this drop target.

**Syntax**

```cpp
void SetOnDropAction(const AZStd::string& actionName)
```

**Tip**

A more flexible way to be notified when a drop occurs is to use the `UiDropTargetNotificationBus (p. 3055).`

**UiDropTargetNotificationBus**

Services notifications for the `UiDropTargetComponent`.

**OnDrop**

Occurs when a draggable component is dropped on this drop target. The draggable component is passed in. Implement the game logic of what should happen on drag and drop here.

**Syntax**

```cpp
void OnDrop(AZ::EntityId draggable)
```

**OnDropHoverEnd**

Occurs when the focus stops being on this drop target during dragging. The draggable component that is being dragged is passed into this function.

**Syntax**

```cpp
void OnDropHoverEnd(AZ::EntityId draggable)
```

**OnDropHoverStart**

Occurs when the focus starts to be on this drop target during dragging. The draggable component that is being dragged is passed into this function.

**Syntax**
void OnDropHoverStart(AZ::EntityId draggable)

**UIDynamicLayoutComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Clones a prototype element to achieve the specified number of child elements.

**UiDynamicLayoutBus**

Services messages for the UiDynamicLayoutComponent.

**SetNumChildElements**

Specifies the number of child elements to be created dynamically.

**Syntax**

```c
void SetNumChildElements(int numChildren)
```

**UIDynamicScrollViewComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Dynamically sets up scrollbox content as a horizontal or vertical list of elements that are cloned from a prototype element. For efficient scrolling, only the minimum number of elements are created.

**UiDynamicScrollViewBus**

Services messages for the UiDynamicScrollViewComponent.

**GetChildElementAtLocationIndex**

Returns the child element at the specified location index.

**Syntax**

```c
AZ::EntityId GetChildElementAtLocationIndex(int index)
```

**GetLocationIndexOfChild**

Returns the location index of the specified child element. Returns -1 if not found.

**Syntax**

```c
int GetLocationIndexOfChild(AZ::EntityId childElement)
```
**RefreshContent**
Refreshes the content. You should call this when the list size or element content has changed.

**Syntax**

```csharp
void RefreshContent()
```

**UiDynamicScrollBoxDataBus**
Create this handler to provide a dynamic scrollbox with the number of elements to display.

**GetNumElements**
Returns the total number of elements that the dynamic scrollbox is to display.

**Syntax**

```csharp
int GetNumElements()
```

**UiDynamicScrollBoxElementNotificationBus**
Create this handler to receive notifications of dynamic scrollbox element state changes, such as when an element is about to scroll into view.

**OnElementBecomingVisible**
An element is about to become visible. Use this event to populate the element with data for display. Specifies the entity ID of the element that is about to become visible and its location index.

**Syntax**

```csharp
void OnElementBecomingVisible(AZ::EntityId entityId, int index)
```

**UIElementComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Provides functionality for UI elements.

**UiElementBus**
Services messages for the UIElementComponent.

**DestroyElement**
Deletes this element and removes it from its parent element.

**Syntax**

```csharp
void DestroyElement()
```
**FindChildByName**

Returns the first immediate child with the specified name. Returns `AZ::InvalidEntityId` if no match is found.

**Syntax**

```
AZ::EntityId FindChildByName(const AZStd::string & name)
```

**FindDescendantByName**

Returns the first descendent entity with the specified name. Returns `AZ::InvalidEntityId` if no match is found.

**Syntax**

```
AZ::EntityId FindDescendantByName(const AZStd::string & name)
```

**GetCanvas**

Returns the canvas that contains this element. Returns `AZ::InvalidEntityId` if the element has no canvas.

**Syntax**

```
AZ::EntityId GetCanvas()
```

**GetChild**

Returns the child entity ID at the specified index. The specified index must be less than `GetNumChildElements()`.

**Syntax**

```
AZ::EntityId GetChild(int index)
```

**GetChildren**

Returns the child entity IDs of this element.

**Syntax**

```
AZStd::vector<AZ::EntityId> GetChildren()
```

**GetIndexOfChildByEntityId**

Returns the index of the specified child element.

**Syntax**

```
int GetIndexOfChildByEntityId(AZ::EntityId childId)
```

**GetName**

Returns the name of this element.
Syntax

AZStd::string GetName()

**GetNumChildElements**

Returns the number of child elements of this element.

Syntax

```cpp
int GetNumChildElements()
```

**GetParent**

Returns the parent entity ID of this element. Returns an invalid entity ID if the element has no parent.

Syntax

```cpp
AZ::EntityId GetParent()
```

**IsAncestor**

Returns whether a specified element is an ancestor of this element.

Syntax

```cpp
bool IsAncestor(AZ::EntityId id)
```

**IsEnabled**

Returns true if the element is enabled; false otherwise.

Syntax

```cpp
bool IsEnabled()
```

**Reparent**

Makes this element the child of a different parent. The element is removed from its current parent and added as a child of the parent specified by `newParent`.

Syntax

```cpp
void Reparent(AZ::EntityId newParent, AZ::EntityId insertBefore)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newParent</code></td>
<td>Specifies the entity ID of the new parent element. If <code>newParent</code> is invalid, the element becomes a top-level element (that is, the canvas becomes the parent).</td>
</tr>
<tr>
<td><code>insertBefore</code></td>
<td>Child element of the new parent before which to insert this element. If null, the element is put at the end of the child list.</td>
</tr>
</tbody>
</table>
**SetIsEnabled**

Sets the enabled state of the element.

**Syntax**

```csharp
void SetIsEnabled(bool isEnabled)
```

**UIFaderComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](https://aws.amazon.com/opendesign/) or visit the AWS Game Tech blog to learn more.

You can use a fader component to simultaneously adjust the transparency of an element and its children.

**UiFaderBus**

Services messages for the UiFaderComponent.

**Fade**

Triggers a fade animation.

**Syntax**

```csharp
void Fade(float targetValue, float speed)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetValue</td>
<td>The value at which to end the fade [0,1].</td>
</tr>
<tr>
<td>speed</td>
<td>The speed of the fade animation in full fade amount per second; 0 means instant.</td>
</tr>
</tbody>
</table>

**GetFadeValue**

Returns the fade value. The fade value is a float between zero and one. One means no fade; zero means complete fade to invisible.

**Syntax**

```csharp
float GetFadeValue()
```

**IsFading**

Returns whether a fade animation is taking place.

**Syntax**

```csharp
bool IsFading()
```
SetFadeValue
Sets the fade value.

**Syntax**

```csharp
void SetFadeValue(float fade)
```

**UiFaderNotificationBus**
Services notifications for the *UiFaderComponent*.

**OnDestroyed**
The fader component has been destroyed.

**Syntax**

```csharp
void OnFaderDestroyed()
```

**OnFadeComplete**
The fade animation is done.

**Syntax**

```csharp
void OnFadeComplete()
```

**OnFadeInterrupted**
The fade animation has been interrupted.

**Syntax**

```csharp
void OnFadeInterrupted()
```

**UIFlipbookAnimationComponent**
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **UIFlipbookAnimationComponent** animates the display of a range of cells in a sprite sheet image. You must use an *Image* component along with the **UIFlipbookAnimationComponent** component.

**UIFlipbookAnimationBus**
Services messages for the **UIFlipbookAnimationComponent**.

**Start**
 Begins playing the flipbook animation.

**Syntax**

```csharp
```
void Start()

**Stop**

Ends the animation.

**Syntax**

```cpp
void End()
```

**IsPlaying**

Returns true if the animation is currently playing. Otherwise, false.

**Syntax**

```cpp
void IsPlaying()
```

**GetStartFrame**

Returns the first frame to display when starting the animation.

**Syntax**

```cpp
AZ::u32 GetStartFrame()
```

**SetStartFrame**

Sets the first frame to display when starting the animation.

**Syntax**

```cpp
void SetStartFrame(AZ::u32 startFrame)
```

**GetEndFrame**

Returns the last frame to display for the animation.

**Syntax**

```cpp
AZ::u32 GetEndFrame()
```

**SetEndFrame**

Sets the last frame to display for the animation.

**Syntax**

```cpp
void SetEndFrame(AZ::u32 endFrame)
```

**GetCurrentFrame**

Returns the frame of the animation currently displayed.
Syntax

AZ::u32 GetCurrentFrame()

**GetCurrentFrame**

Sets the frame to immediately display for the animation.

**Syntax**

void SetCurrentFrame(AZ::u32 currentFrame)

**GetLoopStartFrame**

Returns the first frame that is displayed within an animation loop.

Applicable only when the **Loop Type** is set to anything other than **None**.

**Syntax**

AZ::u32 GetLoopStartFrame()

**SetLoopStartFrame**

Sets the first frame that is displayed within an animation loop.

Applicable only when the **Loop Type** is set to anything other than **None**.

**Syntax**

void SetLoopStartFrame(AZ::u32 loopStartFrame)

**GetLoopType**

Returns the type of looping behavior for this animation.

**Syntax**

eUiFlipbookAnimationLoopType GetLoopType()

See **SetLoopType** for a list of possible loop type values.

**SetLoopType**

Sets the type of looping behavior for this animation.

**Syntax**

void SetLoopType(eUiFlipbookAnimationLoopType loopType)

Possible loop types are as follows:

```cpp
eUiFlipbookAnimationLoopType
{
...
Each type performs a different looping action:

- **None** – Animation ends when end frame is reached.
- **Linear** – The frame displayed after the end frame is always the Loop Start Frame.
- **PingPong** – The direction of the animation loop goes back and forth between the start frame and the end frame.

**GetFrameDelay**

Returns the number of seconds to wait before displaying the next frame.

**Syntax**

```cpp
float GetFrameDelay()
```

**SetFrameDelay**

Sets the number of seconds to wait before displaying the next frame.

**Syntax**

```cpp
void SetFrameDelay(float delay)
```

**GetIsAutoPlay**

Returns true if the animation will begin playing as soon as the element is activated. Otherwise, false.

**Syntax**

```cpp
bool GetIsAutoPlay()
```

**SetIsAutoPlay**

Sets whether the animation will begin playing as soon as the element is activated.

**Syntax**

```cpp
void SetIsAutoPlay(bool isAutoPlay)
```

**UiFlipbookAnimationNotificationBus**

Notifies listeners of important events concerning the UiFlipbookAnimationComponent.

**OnAnimationStarted**

The flipbook animation has begun playing.

**Syntax**
void OnAnimationStarted()

**OnAnimationStopped**

The flipbook animation has stopped playing.

**Syntax**

```c++
void OnAnimationStopped()
```

**OnLoopSequenceCompleted**

The flipbook animation has completed one loop iteration. This triggers only when the Loop Type of the flipbook animation is configured to anything other than None.

For **Linear** loops, this triggers when End Frame is displayed.

For **Ping Pong** loops, this triggers when either Start Frame or End Frame is displayed (depending on the current loop direction of the loop).

**Syntax**

```c++
void OnLoopSequenceCompleted()
```

**UIImageComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls sprites, images, and textures.

**UIImageBus**

Services messages for the UiImageComponent.

**GetColor**

Returns the color tint for the image.

**Syntax**

```c++
AZ::Color GetColor()
```

**SetColor**

Sets the color tint for the image.

**Syntax**

```c++
void SetColor(const AZ::Color& color)
```
GetSpritePathname

Returns the source location of the image to be displayed by the element.

Syntax

```azstd::string GetSpritePathname()```

SetSpritePathname

Sets the source location of the image to be displayed by the element.

Syntax

```void SetSpritePathname(AZStd::string spritePath)```

GetRenderTargetName

Returns the name of the render target associated with the sprite.

Syntax

```azstd::string GetRenderTargetName()```

SetRenderTargetName

Sets the name of the render target associated with the sprite.

Syntax

```void SetRenderTargetName(AZStd::string renderTargetName)```

GetSpriteType

Returns the type of the sprite.

Syntax

```eUiSpriteType GetSpriteType()```

Possible sprite types are as follows.

```enum eUiSpriteType
    {
        eUiSpriteType_SpriteAsset,
        eUiSpriteType_RenderTarget
    }```

SetSpriteType

Sets the type of the sprite.

Syntax
void SetSpriteType(eUiSpriteType spriteType)

For possible sprite types, see GetSpriteType (p. 3066).

**GetImageType**

Returns the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

**Syntax**

```cpp
eUiImageType GetImageType()
```

Possible values for `eUiImageType` are as follows.

```cpp
enum eUiImageType
{
    eUiImageType_Stretched,
    eUiImageType_Sliced,
    eUiImageType_Fixed,
    eUiImageType_Tiled,
    eUiImageType_StretchedToFit,
    eUiImageType_StretchedToFill
};
```

**SetImageType**

Sets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

**Syntax**

```cpp
void SetImageType(eUiImageType imageType)
```

For possible values for `eUiImageType`, see GetImageType (p. 3067).

**GetFillType**

Returns the Fill Type of the image. Fill Type determines how the image component is filled.

**Syntax**

```cpp
eUiFillType GetFillType()
```

Possible values for `eUiFillType` are as follows.

```cpp
enum eUiFillType
{
    eUiFillType_None,
    eUiFillType_Linear,
    eUiFillType_Radial,
    eUiFillType_RadialCorner,
    eUiFillType_RadialEdge
};
```

**SetFillType**

Sets the Fill Type of the image. Fill Type determines how the image component is filled.
Syntax

```
void SetFillType(eUiFillType fillType)
```

For possible values for `eUiFillType`, see `GetFillType` (p. 3067).

**GetFillAmount**

Returns the **Fill Amount**. The **Fill Amount** is a float between zero and one. 1.00 indicates that the image is completely filled. 0.00 means no part of the image is filled.

Syntax

```
float GetFillAmount()
```

**SetFillAmount**

Sets the **Fill Amount**.

Syntax

```
void SetFillAmount(float fillAmount)
```

**GetRadialFillStartAngle**

Returns the starting angle of the **Radial Fill** in degrees clockwise. A value of 0 indicates the top center of the image.

Syntax

```
float GetRadialFillStartAngle()
```

**SetRadialFillStartAngle**

Sets the starting angle of the **Radial Fill**.

Syntax

```
void SetRadialFillStartAngle(float radialFillStartAngle)
```

**GetCornerFillOrigin**

Returns the **Corner Fill** origin of the image.

Syntax

```
eUiFillCornerOrigin GetCornerFillOrigin()
```

Possible values for `eUiFillCornerOrigin` are as follows.

```
enum eUiFillCornerOrigin
{
    eUiFillCornerOrigin_TopLeft,
    eUiFillCornerOrigin_TopRight,
    // other values...
}
```
SetCornerFillOrigin

Sets the **Corner Fill** origin of the image.

**Syntax**

```c
void SetCornerFillOrigin(eUiFillCornerOrigin cornerFillOrigin)
```

For possible values for `eUiFillCornerOrigin`, see [GetCornerFillOrigin](#) (p. 3068).

GetEdgeFillOrigin

Returns the **Edge Fill** origin of the image.

**Syntax**

```c
enum eUiFillEdgeOrigin GetEdgeFillOrigin()
```

Possible values for `eUiFillEdgeOrigin` are as follows.

```c
enum eUiFillEdgeOrigin
{
    eUiFillEdgeOrigin_Left,
    eUiFillEdgeOrigin_Top,
    eUiFillEdgeOrigin_Right,
    eUiFillEdgeOrigin_Bottom
};
```

SetEdgeFillOrigin

Sets the **Edge Fill** origin of the image.

**Syntax**

```c
void SetEdgeFillOrigin(eUiFillEdgeOrigin edgeFillOrigin)
```

For possible values for `eUiFillEdgeOrigin`, see [GetEdgeOrigin](#) (p. 3069).

GetFillClockwise

Returns whether the image is radially filled clockwise.

**Syntax**

```c
bool GetFillClockwise()
```

SetFillClockwise

Sets whether the image is radially filled clockwise.

**Syntax**

```c
```
void SetFillClockwise(bool fillClockwise)

**GetFillCenter**

Returns whether the center of a sliced image is filled.

**Syntax**

```cpp
bool GetFillCenter()
```

**SetFillCenter**

Sets whether the center of a sliced image is filled.

**Syntax**

```cpp
void SetFillCenter(bool fillCenter)
```

**SetSpriteSheetCellIndex**

Sets the sprite sheet cell index for the image component to display.

**Syntax**

```cpp
void SetSpriteSheetCellIndex(AZ::u32 index)
```

**GetSpriteSheetCellIndex**

Returns the sprite sheet cell index currently displayed by the image component.

**Syntax**

```cpp
AZ::u32 GetSpriteSheetCellIndex()
```

**GetSpriteSheetCellCount**

Returns the number of cells in the sprite sheet.

If no image file is associated with this component, returns 0.

If an image file is associated but not configured as a sprite sheet, returns 1.

**Syntax**

```cpp
AZ::u32 GetSpriteSheetCellCount()
```

**GetSpriteSheetCellAlias**

Returns the string alias associated with the given sprite sheet cell index.

Returns an empty string if an alias is not configured or is unavailable.

**Syntax**
AZStd::string GetSpriteSheetCellAlias(AZ::u32 index)

**SetSpriteSheetCellAlias**

Given a cell index that is valid for the sprite sheet image associated with the image component, assign the alias string for that cell.

**Syntax**

```cpp
void SetSpriteSheetCellAlias(AZ::u32 index, AZStd::string alias)
```

**GetSpriteSheetCellIndexFromAlias**

Given a string alias that corresponds to an alias of a cell within the sprite sheet image of the component, return that cells index value.

**Note**

This returns only the first matching cell in the sprite sheet that matches the given string. Other cells in the sprite sheet that have the same alias are not considered after the first match is found.

**Syntax**

```cpp
AZ::u32 GetSpriteSheetCellIndexFromAlias(AZStd::string alias)
```

**UIInteractableComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls elements that respond to user input.

**UIInteractableBus**

Services messages for the UIInteractableComponent.

**GetIsAutoActivationEnabled**

Returns true if automatic activation is enabled; false otherwise.

**Syntax**

```cpp
bool GetIsAutoActivationEnabled()
```

**IsHandlingEvents**

Returns true if event handling is enabled; false otherwise.

**Syntax**

```cpp
bool IsHandlingEvents()
```
**SetIsAutoActivationEnabled**
Sets whether automatic activation is enabled.

**Syntax**

```cpp
void SetIsAutoActivationEnabled (bool isEnabled)
```

**SetIsHandlingEvents**
Sets whether event handling is enabled.

**Syntax**

```cpp
void SetIsHandlingEvents (bool isHandlingEvents)
```

**UiInteractableActionsBus**
Services actions for the UiInteractableComponent.

**GetHoverEndActionName**
Returns the current hover end action name.

**Syntax**

```cpp
AZStd::string& GetHoverEndActionName()
```

**GetHoverStartActionName**
Returns the current hover start action name.

**Syntax**

```cpp
AZStd::string& GetHoverStartActionName()
```

**GetPressedActionName**
Returns the pressed action name.

**Syntax**

```cpp
AZStd::string& GetPressedActionName()
```

**GetReleasedActionName**
Returns the released action name.

**Syntax**

```cpp
AZStd::string& GetReleasedActionName()
```

**SetHoverEndActionName**
Sets the hover end action name.
Syntax

SetHoverEndActionName(const AZStd::string& actionName)

**SetHoverStartActionName**
Sets the hover start action name.

**Syntax**

SetHoverStartActionName(const AZStd::string& actionName)

**SetPressedActionName**
Sets the pressed action name.

**Syntax**

SetPressedActionName(const AZStd::string& actionName)

**SetReleasedActionName**
Sets the released action name.

**Syntax**

SetReleasedActionName(const AZStd::string& actionName)

**UiInteractableStatesBus**
Services states for the UiInteractableComponent.

**GetStateAlpha**
Returns the alpha to be used for the specified target when the interactable element is in the specified state.

**Syntax**

float GetStateAlpha(eUiInteractableState state, AZ::EntityId target)

**GetStateColor**
Returns the color to be used for the specified target when the interactable element is in the specified state.

**Syntax**

AZ::Color GetStateColor(eUiInteractableState state, AZ::EntityId target)

Possible values for eUiInteractableState are as follows.

```cpp
gen eUiInteractableState
```
UIInteractableComponent

```cpp
{
    eUiInteractableState_Normal,
    eUiInteractableState_Hover,
    eUiInteractableState_Pressed,
    eUiInteractableState_Disabled
};
```

**GetStateFontEffectIndex**

Returns the font effect to be used for the specified target when the interactable element is in the specified state.

**Syntax**

```cpp
unsigned int GetStateFontEffectIndex(eUiInteractableState state, AZ::EntityId target)
```

**GetStateFontPathname**

Returns the font path to be used for the specified target when the interactable element is in the specified state.

**Syntax**

```cpp
AZStd::string GetStateFontPathname(eUiInteractableState state, AZ::EntityId target)
```

**GetStateSpritePathname**

Returns the sprite path to be used for the specified target when the interactable element is in the specified state.

**Syntax**

```cpp
AZStd::string GetStateSpritePathname(eUiInteractableState state, AZ::EntityId target)
```

**HasStateAlpha**

Returns true if the interactable element has an alpha action for the specified state and target combination.

**Syntax**

```cpp
bool HasStateAlpha(eUiInteractableState state, AZ::EntityId target)
```

**HasStateColor**

Returns true if the interactable element has a color action for the specified state and target combination.

**Syntax**

```cpp
bool HasStateColor(eUiInteractableState state, AZ::EntityId target)
```

**HasStateFont**

Returns true if the interactable element has a font action for the specified state and target combination.
UIInteractableComponent

Syntax

```cpp
bool HasStateFont(eUiInteractableState state, AZ::EntityId target)
```

**HasStateSprite**

Returns true if the interactable element has a sprite action for the specified state and target combination.

Syntax

```cpp
bool HasStateSprite(eUiInteractableState state, AZ::EntityId target)
```

**SetStateAlpha**

Sets the alpha to be used for the specified target when the interactable element is in the specified state. If the interactable element already has an alpha action for this state and target combination, then SetStateAlpha replaces the alpha action.

Syntax

```cpp
void SetStateAlpha(eUiInteractableState state, AZ::EntityId target, float alpha)
```

**SetStateColor**

Sets the color to be used for the specified target when the interactable element is in the specified state. If the interactable element already has a color action for the current state and target combination, then SetStateColor replaces the color action.

Syntax

```cpp
void SetStateColor(eUiInteractableState state, AZ::EntityId target, const AZ::Color& color)
```

**SetStateFont**

Sets the font to be used for the specified target when the interactable element is in the specified state. If the interactable element already has a font action for the specified state and target combination, then SetStateFont replaces the font action.

Syntax

```cpp
void SetStateFont(eUiInteractableState state, AZ::EntityId target, const AZStd::string& fontPathname, unsigned int fontEffectIndex)
```

**SetStateSpritePathname**

Sets the sprite path to be used for the specified target when the interactable element is in the specified state. If the interactable element already has a sprite action for the specified state and target combination, then SetStateSpritePathname replaces the sprite action.

Syntax

```cpp
void SetStateSpritePathname(eUiInteractableState state, AZ::EntityId target, const AZStd::string& spritePath)
```
UiInteractableNotificationBus
Services notifications for the UIInteractableComponent.

OnHoverEnd
Called on hover end.
Syntax

void OnHoverEnd()

OnHoverStart
Called on hover start.
Syntax

void OnHoverStart()

OnPressed
Called when an element has been pressed.
Syntax

void OnPressed()

OnReleased
Called when an element has been released.
Syntax

void OnReleased()

OnReceivedHoverByNavigatingFromDescendant
Called when the element receives the hover by being navigated to from a descendant.
Syntax

void OnReceivedHoverByNavigatingFromDescendant(AZ::EntityId descendantEntityId)

UiNavigationBus
Services navigation for the UIInteractableComponent.

GetNavigationMode
Returns the navigation mode.
Syntax
eUiNavigationMode GetNavigationMode()

Possible values for eUiNavigationMode are as follows.

enum eUiNavigationMode
{
    eUiNavigationMode_Automatic,
    eUiNavigationMode_Custom,
    eUiNavigationMode_None
};

GetOnDownEntity

Returns the ID of the entity that receives focus when down is pressed.

Syntax

AZ::EntityId GetOnDownEntity()

GetOnLeftEntity

Returns the ID of the entity to receive focus when left is pressed.

Syntax

AZ::EntityId GetOnLeftEntity()

GetOnRightEntity

Returns the ID of the entity to receive focus when right is pressed.

Syntax

AZ::EntityId GetOnRightEntity()

GetOnUpEntity

Returns the ID of the entity that receives focus when up is pressed.

Syntax

AZ::EntityId GetOnUpEntity()

SetNavigationMode

Sets the navigation mode.

Syntax

void SetNavigationMode(eUiNavigationMode navigationMode)

SetOnDownEntity

Sets the entity to receive focus when down is pressed.
Syntax

```cpp
void SetOnDownEntity(AZ::EntityId entityId)
```

### SetOnLeftEntity

Sets the entity to receive focus when left is pressed.

**Syntax**

```cpp
void SetOnLeftEntity(AZ::EntityId entityId)
```

### SetOnRightEntity

Sets the entity to receive focus when right is pressed.

**Syntax**

```cpp
void SetOnRightEntity(AZ::EntityId entityId)
```

### SetOnUpEntity

Sets the entity to receive focus when up is pressed.

**Syntax**

```cpp
void SetOnUpEntity(AZ::EntityId entityId)
```

## UILayout

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Provides common functionality for row, column, and grid layouts.

### UILayoutBus

Services messages for the UILayout components.

### GetHorizontalChildAlignment

Returns the horizontal child alignment.

**Syntax**

```cpp
eUiHAlign GetHorizontalChildAlignment()
```

Following are possible values for `eUiHAlign`.

```cpp
enum eUiHAlign
{
    eUiHAlign_Left,
    eUiHAlign_Center,
}```
GetIgnoreDefaultLayoutCells

Returns whether default layout cell values calculated by other components on the child are ignored.

Syntax

```cpp
bool GetIgnoreDefaultLayoutCells()
```

GetVerticalChildAlignment

Returns the vertical child alignment.

Syntax

```cpp
eUiVAlign GetVerticalChildAlignment()
```

Following are possible values for `eUiVAlign`.

```cpp
class enum eUiVAlign
{
    eUiVAlign_Top,
    eUiVAlign_Center,
    eUiVAlign_Bottom
};
```

SetHorizontalChildAlignment

Sets the horizontal child alignment.

Syntax

```cpp
void SetHorizontalChildAlignment(eUiHAlign hAlign)
```

For possible values for `eUiHAlign`, see `GetHorizontalChildAlignment (p. 3078)`.

SetIgnoreDefaultLayoutCells

Sets whether default layout cell values calculated by other components on the child are ignored.

Syntax

```cpp
void SetIgnoreDefaultLayoutCells (bool ignore)
```

SetVerticalChildAlignment

Sets the vertical child alignment.

Syntax

```cpp
void SetVerticalChildAlignment(eUiVAlign vAlign)
```

For possible values for `eUiVAlign`, see `GetVerticalChildAlignment (p. 3079)`. 
UiLayoutCellComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Overrides default layout cell values.

UiLayoutCellBus

Services messages for the UiLayoutCellComponent.

GetExtraHeightRatio

Returns the overridden extra height ratio for the element. A value of –1 means that the ratio has not been overridden.

Syntax

```c
float GetExtraHeightRatio()
```

GetExtraWidthRatio

Returns the overridden extra width ratio for the element. A value of –1 means that the ratio has not been overridden.

Syntax

```c
float GetExtraWidthRatio()
```

GetMinHeight

Returns the overridden minimum height for the element. A value of –1 means that the height has not been overridden.

Syntax

```c
float GetMinHeight()
```

GetMinWidth

Returns the overridden minimum width for the element. A value of –1 means it has not been overridden.

Syntax

```c
float GetMinWidth()
```

GetTargetHeight

Returns the overridden target height for the element. A value of –1 means that the height has not been overridden.

Syntax

```c
```
float GetTargetHeight()

**GetTargetWidth**

Returns the overridden target width for the element. -1 means it has not been overridden.

**Syntax**

```c
float GetTargetWidth()
```

**SetExtraHeightRatio**

Sets the overridden extra height ratio for the element. A value of -1 means don’t override.

**Syntax**

```c
void SetExtraHeightRatio(float height)
```

**SetExtraWidthRatio**

Sets the overridden extra width ratio for the element. A value of -1 means don’t override.

**Syntax**

```c
void SetExtraWidthRatio(float width)
```

**SetMinHeight**

Sets the overridden minimum height for the element. A value of -1 means don’t override.

**Syntax**

```c
void SetMinHeight(float height)
```

**SetMinWidth**

Sets the overridden minimum width for the element. A value of -1 means don’t override.

**Syntax**

```c
void SetMinWidth(float width)
```

**SetTargetHeight**

Sets the overridden target height for the element. A value of -1 means don’t override.

**Syntax**

```c
void SetTargetHeight(float height)
```

**SetTargetWidth**

Sets the overridden target width for the element. A value of -1 means don’t override.
Syntax

```c
void SetTargetWidth(float width)
```

**UILayoutColumnComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls column layout.

**UILayoutColumnBus**

Services messages for the **UILayoutColumnComponent**.

**GetOrder**

Returns the vertical order for this layout.

**Syntax**

```c
eUiVerticalOrder GetOrder()
```

Following are the possible values for **eUiVerticalOrder**.

```c
enum eUiVerticalOrder
{
    eUiVerticalOrder_TopToBottom,
    eUiVerticalOrder_BottomToTop
};
```

**GetPadding**

Returns, in pixels, the padding inside the edges of the element.

**Syntax**

```c
UiPadding GetPadding()
```

Following are the possible values for **UiPadding**.

```c
class UiPadding
{
    int left;
    int right;
    int top;
    int bottom;
};
```

**GetSpacing**

Returns, in pixels, the spacing between child elements.
Syntax

```plaintext
float GetSpacing()
```

**SetOrder**

Sets the vertical order for this layout.

**Syntax**

```plaintext
void SetOrder(eUiVerticalOrder order)
```

For possible values for `eUiVerticalOrder`, see `GetOrder (p. 3082)`.

**SetPadding**

Sets the padding inside the edges of the element to the pixel value specified.

**Syntax**

```plaintext
void SetPadding(UiPadding padding)
```

For the possible values for `UiPadding`, see `GetPadding (p. 3082)`.

**SetSpacing**

Sets the spacing between child elements to the pixel value specified.

**Syntax**

```plaintext
void SetSpacing(float spacing)
```

**UILayoutFitterComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use this component to implement a UI Layout Fitter in Amazon Lumberyard.

**UILayoutFitterBus**

Services messages for the `UILayoutFitterComponent`.

**GetHorizontalFit**

Returns whether to resize the element horizontally.

**Syntax**

```plaintext
bool GetHorizontalFit()
```
SetHorizontalFit
Sets whether to resize the element horizontally.

Syntax

```csharp
void SetHorizontalFit(bool horizontalFit)
```

GetVerticalFit
Returns whether to resize the element vertically.

Syntax

```csharp
bool GetVerticalFit()
```

SetVerticalFit
Sets whether to resize the element vertically.

Syntax

```csharp
void SetVerticalFit(bool verticalFit)
```

UILayoutGridComponent
Controls grid layout characteristics.

UiLayoutGridBus
Services messages for the UiLayoutGridComponent.

GetCellSize
Returns, in pixels, the size of a child element.

Syntax

```csharp
AZ::Vector2 GetCellSize()
```

GetHorizontalOrder
Returns the horizontal order for the layout.

Syntax

```csharp
eUiHorizontalOrder GetHorizontalOrder()
```
Following are possible values for `eUiHorizontalOrder`.

```csharp
enum eUiHorizontalOrder
{
    eUiHorizontalOrder_LeftToRight,
    eUiHorizontalOrder_RightToLeft
};
```

**GetPadding**

Returns, in pixels, the padding inside the edges of the element.

**Syntax**

```csharp
UiPadding GetPadding()
```

Following are the attributes for `UiPadding`.

```csharp
class UiPadding
{
    int left;
    int right;
    int top;
    int bottom;
};
```

**GetSpacing**

Returns, in pixels, the spacing between child elements.

**Syntax**

```csharp
AZ::Vector2 GetSpacing()
```

**GetStartingDirection**

Returns the starting direction for the layout.

**Syntax**

```csharp
eUiLayoutGridStartingDirection GetStartingDirection()
```

Following are possible values for `eUiLayoutGridStartingDirection`.

```csharp
enum eUiLayoutGridStartingDirection
{
    eUiLayoutGridStartingDirection_HorizontalOrder,
    eUiLayoutGridStartingDirection_VerticalOrder
};
```

**GetVerticalOrder**

Returns the vertical order for the layout.

**Syntax**
eUiVerticalOrder GetVerticalOrder()

Following are possible values for eUiVerticalOrder.

enum eUiVerticalOrder

{ eUiVerticalOrder_TopToBottom,
  eUiVerticalOrder_BottomToTop
};

**SetCellSize**

Sets the size of a child element to the specified number of pixels.

**Syntax**

```c
void SetCellSize(AZ::Vector2 size)
```

**SetHorizontalOrder**

Sets the horizontal order for the layout.

**Syntax**

```c
void SetHorizontalOrder(eUiHorizontalOrder order)
```

For possible values for eUiHorizontalOrder, see GetHorizontalOrder (p. 3084).

**SetPadding**

Sets the padding inside the edges of the element to the specified number of pixels.

**Syntax**

```c
void SetPadding(UiPadding padding)
```

For possible values for UiPadding, see GetPadding (p. 3085).

**SetSpacing**

Sets the spacing between child elements to the specified number of pixels.

**Syntax**

```c
void SetSpacing(AZ::Vector2 spacing)
```

**SetStartingDirection**

Sets the starting direction for the layout.

**Syntax**

```c
void SetStartingDirection(eUiLayoutGridStartingDirection direction)
```
For possible values for `eUiLayoutGridStartingDirection`, see `GetStartingDirection (p. 3085)`.

**SetVerticalOrder**

Sets the vertical order for the layout.

**Syntax**

```cpp
void SetVerticalOrder(eUiVerticalOrder order)
```

For possible values for `eUiVerticalOrder`, see `GetVerticalOrder (p. 3085)`.

**UILayoutRowComponent**

Controls the grid layout of rows.

**UILayoutRowBus**

Messages serviced by the `UILayoutRowComponent`.

**GetOrder**

Returns the horizontal order for this layout.

**Syntax**

```cpp
eUiHorizontalOrder GetOrder()
```

Following are possible values for `eUiHorizontalOrder`.

```cpp
eUiHorizontalOrder
{
    eUiHorizontalOrder_LeftToRight,
    eUiHorizontalOrder_RightToLeft
};
```

**GetPadding**

Returns, in pixels, the padding inside the edges of the element.

**Syntax**

```cpp
UiPadding GetPadding()
```

Following are possible values for `UiPadding`.

```cpp
class UiPadding
{
    int left;
};
```
int right;
int top;
int bottom;
);

GetSpacing

Returns, in pixels, the spacing between child elements.

Syntax

float GetSpacing()

SetOrder

Sets the horizontal order for this layout.

Syntax

void SetOrder(eUiHorizontalOrder order)

For possible values for eUiHorizontalOrder, see GetOrder (p. 3087).

SetPadding

Sets the padding inside the edges of the element to the specified number of pixels.

Syntax

void SetPadding(UiPadding padding)

For possible values for UiPadding, see GetPadding (p. 3087).

SetSpacing

Sets the spacing between child elements to the specified number of pixels.

Syntax

void SetSpacing(float spacing)

UIMaskComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can add a mask component to an element to show only a part of the content of its child elements (for example, example, image or text).

UIMaskBus

Services messages for the UIMaskComponent.
GetDrawBehind
Returns whether the mask is drawn behind the child elements.

Syntax

```cpp
bool GetDrawBehind()
```

GetDrawInFront
Returns whether the mask is drawn in front of child elements.

Syntax

```cpp
bool GetDrawInFront()
```

GetIsMaskingEnabled
Returns whether masking is enabled.

Syntax

```cpp
bool GetIsMaskingEnabled()
```

GetUseAlphaTest
Returns whether to use the alpha channel in the mask visual's texture to define the mask.

Syntax

```cpp
bool GetUseAlphaTest()
```

SetDrawBehind
Sets whether the mask is drawn behind the child elements.

Syntax

```cpp
void SetDrawBehind(bool drawMaskVisualBehindChildren)
```

SetDrawInFront
Sets whether the mask is drawn in front of child elements.

Syntax

```cpp
void SetDrawInFront(bool drawMaskVisualInFrontOfChildren)
```

SetIsMaskingEnabled
Sets whether masking is enabled.

Syntax

```cpp
```
void SetIsMaskingEnabled(bool enableMasking)

**SetUseAlphaTest**

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

**Syntax**

void SetUseAlphaTest(bool useAlphaTest)

**UIParticleEmitterComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls the emission of two dimensional particles. The source code location is `\dev\Gems\LyShine\Code\Source\UiParticleEmitterComponent.*`.

**UIParticleEmitterBus**

Services messages for the `UIParticleEmitterComponent`.

**GetIsEmitting**

Returns whether the emitter is currently emitting.

**Syntax**

bool GetIsEmittingColor()

**SetIsEmitting**

Sets whether the emitter is currently emitting.

**Syntax**

void SetIsEmitting(bool isEmitting)

**GetIsRandomSeedFixed**

Returns whether the emitter uses a fixed random seed.

**Syntax**

bool GetIsRandomSeedFixed()

**SetIsRandomSeedFixed**

Sets whether the emitter uses a fixed random seed.

**Syntax**
void SetRandomSeed(bool randomSeedFixed)

**GetRandomSeed**

Returns the current random seed.

**Syntax**

```c
int GetRandomSeed()
```

**SetRandomSeed**

Sets the random seed used by the emitter.

**Syntax**

```c
void SetRandomSeed(int randomSeed)
```

**GetIsParticlePositionRelativeToEmitter**

Returns whether the emitted particles move relative to the emitter.

**Syntax**

```c
bool GetIsParticlePositionRelativeToEmitter()
```

**SetIsParticlePositionRelativeToEmitter**

Sets whether the emitted particles move relative to the emitter.

**Syntax**

```c
void SetIsParticlePositionRelativeToEmitter(bool relativeToEmitter)
```

**GetParticleEmitRate**

Returns, in particles per second, the current particle emitter emit rate.

**Syntax**

```c
float GetParticleEmitRate()
```

**SetParticleEmitRate**

Sets, in particles per second, the current particle emitter emit rate.

**Syntax**

```c
void SetParticleEmitRate(float particleEmitRate)
```

**GetIsEmitOnActivate**

Returns whether the particle emitter starts emitting when the component is activated.
Syntax

bool GetIsEmitOnActivate()

**SetIsEmitOnActivate**

Sets whether the particle emitter starts emitting when the component is activated.

Syntax

void SetIsEmitOnActivate(bool emitOnActivate)

**GetIsHitParticleCountOnActivate**

Returns whether the average amount of particles are emitted and processed when the emitter starts emitting.

Syntax

bool GetIsHitParticleCountOnActivate()

**SetIsHitParticleCountOnActivate**

Sets whether the average amount of particles are emitted and processed when the emitter starts emitting.

Syntax

void SetIsHitParticleCountOnActivate(bool hitParticleCountOnActivate)

**GetIsEmitterLifetimeInfinite**

Returns whether the emitter lifetime is infinite.

Syntax

bool GetIsEmitterLifetimeInfinite()

**SetIsEmitterLifetimeInfinite**

Sets whether the emitter lifetime is infinite.

Syntax

void SetIsEmitterLifetimeInfinite(bool emitterLifetimeInfinite)

**GetEmitterLifetime**

Returns the total emitter lifetime in seconds. When the lifetime is reached, the emitter stops emitting.

Syntax

float GetEmitterLifetime()
**SetEmitterLifetime**

Sets the total emitter lifetime in seconds. When the lifetime is reached, the emitter stops emitting.

**Syntax**

```c
void SetEmitterLifetime(float emitterLifetime)
```

**GetIsParticleCountLimited**

Returns whether there is a limit to the number of active particles.

**Syntax**

```c
bool GetIsParticleCountLimited()
```

**SetIsParticleCountLimited**

Sets whether there is a limit to the number of active particles.

**Syntax**

```c
void SetIsParticleCountLimited(bool particleCountLimited)
```

**GetMaxParticles**

Returns the numerical limit of active particles.

**Syntax**

```c
AZ::u32 GetMaxParticles()
```

**SetMaxParticles**

Sets the numerical limit of active particles.

**Syntax**

```c
void SetMaxParticles(AZ::u32 maxParticles)
```

**GetEmitterShape**

Returns the current emitter shape.

**Syntax**

```c
eUiEmitShape GetEmitterShape()
```

Possible emitter shapes are as follows.

```c
enum eUiEmitShape
{
    eUiEmitShape_Point,
    // Other shapes...
}
```
SetEmitterShape

Sets the current emitter shape.

Syntax

```c
void SetVariable(eUiEmitShape emitShape)
```

For possible emitter shapes, see GetEmitterShape (p. 3093).

GetIsEmitOnEdge

Returns whether the particles are emitted on the edge of the selected shape.

Syntax

```c
bool GetIsEmitOnEdge()
```

SetIsEmitOnEdge

Sets whether the particles are emitted on the edge of the selected shape.

Syntax

```c
void SetIsEmitOnEdge(bool emitOnEdge)
```

GetInsideEmitDistance

Returns the distance that particles are emitted inside the emitter shape edge.

Syntax

```c
float GetInsideEmitDistance()
```

SetInsideEmitDistance

Sets the distance that particles are emitted inside the emitter shape edge.

Syntax

```c
void SetInsideEmitDistance(float insideEmitDistance)
```

GetOutsideEmitDistance

Returns the distance that particles are emitted outside the emitter shape edge.

Syntax

```c
float GetOutsideEmitDistance()
```
**SetOutsideEmitDistance**

Sets the distance that particles are emitted outside the emitter shape edge.

**Syntax**

```cpp
void SetOutsideEmitDistance(float outsideEmitDistance)
```

**GetParticleInitialDirectionType**

Returns how the initial direction of the emitted particles are calculated.

**Syntax**

```cpp
eUiParticleInitialDirectionType GetParticleInitialDirectionType()
```

Possible values are as follows.

```cpp
enum eUiParticleInitialDirectionType
{
    eUiParticleInitialDirectionType_RelativeToEmitAngle,
    eUiParticleInitialDirectionType_RelativeToEmitterCenter
};
```

**SetParticleInitialDirectionType**

Sets how the initial direction of the emitted particles is calculated.

**Syntax**

```cpp
void SetVariable(eUiEmitShape emitShape)
```

For possible direction types, see [GetParticleInitialDirectionType](#) (p. 3095).

**GetEmitAngle**

Returns, in degrees clockwise from straight up, the angle along which particles are emitted.

**Syntax**

```cpp
float GetEmitAngle()
```

**SetEmitAngle**

Sets, in degrees clockwise from straight up, the angle along which particles are emitted.

**Syntax**

```cpp
void SetEmitAngle(float emitAngle)
```

**GetEmitAngleVariation**

Returns, in degrees, the variation in the emit angle. For example, a variation value of 10 designates a range of plus or minus 10 degrees on each side of the current emit angle.
Syntax

float GetEmitAngleVariation()

SetEmitAngleVariation

Sets, in degrees, the variation in the emit angle. For example, a variation value of 10 designates a range of plus or minus 10 degrees on each side of the current emit angle.

Syntax

void SetEmitAngleVariation(float emitAngleVariation)

GetIsParticleLifetimeInfinite

Returns whether the emitted particles have an infinite lifetime.

Syntax

bool GetIsParticleLifetimeInfinite()

SetIsParticleLifetimeInfinite

Sets whether the emitted particles have an infinite lifetime.

Syntax

void SetIsParticleLifetimeInfinite(bool infiniteLifetime)

GetParticleLifetime

Returns, in seconds, the lifetime of the emitted particles.

Syntax

float GetParticleLifetime()

SetParticleLifetime

Sets, in seconds, the initial lifetime of the emitted particles.

Syntax

void SetParticleLifetime(float lifetime)

GetParticleLifetimeVariation

Returns the variation in lifetime of the emitted particles. For example, a variation of 5 seconds designates a range of 5 seconds on either side of the chosen initial lifetime.

Syntax

float GetParticleLifetimeVariation()
**SetParticleLifetimeVariation**

Sets the variation in lifetime of the emitted particles. For example, a variation of 5 seconds designates a range of 5 seconds on either side of the chosen initial lifetime.

**Syntax**

```cpp
void SetParticleLifetimeVariation(float lifetimeVariation)
```

**GetSpritePathname**

Returns the source location of the image to be used by the emitted particles.

**Syntax**

```cpp
AZStd::string GetSpritePathname()
```

**SetSpritePathname**

Sets the source location of the image to be used by the emitted particles.

**Syntax**

```cpp
void SetSpritePathname(AZStd::string spritePath)
```

**GetIsSpriteSheetAnimated**

Returns whether the sprite sheet cell index changes over time on each particle.

**Syntax**

```cpp
bool GetIsSpriteSheetAnimated()
```

**SetIsSpriteSheetAnimated**

Sets whether the sprite sheet cell index changes over time on each particle.

**Syntax**

```cpp
void SetIsSpriteSheetAnimated(bool spriteSheetAnimated)
```

**GetIsSpriteSheetAnimationLooped**

Returns whether the sprite sheet cell animation is looped.

**Syntax**

```cpp
bool GetIsSpriteSheetAnimationLooped()
```

**SetIsSpriteSheetAnimationLooped**

Sets whether the sprite sheet cell animation is looped.
Syntax

```c
void SetIsSpriteSheetAnimationLooped(bool spriteSheetAnimationLooped)
```

**GetIsSpriteSheetIndexRandom**

Returns whether the initial sprite-sheet index is randomly chosen.

Syntax

```c
bool GetIsSpriteSheetIndexRandom()
```

**SetIsSpriteSheetIndexRandom**

Sets whether the initial sprite-sheet index is randomly chosen.

Syntax

```c
void SetIsSpriteSheetIndexRandom(bool spriteSheetIndexRandom)
```

**GetSpriteSheetCellIndex**

Returns the sprite-sheet index to be used for emitted particles.

Syntax

```c
int GetSpriteSheetCellIndex()
```

**SetSpriteSheetCellIndex**

Sets the sprite-sheet index to be used for emitted particles.

Syntax

```c
void SetSpriteSheetCellIndex(int spriteSheetIndex)
```

**GetSpriteSheetCellEndIndex**

Returns the end index of the sprite-sheet cell range that is used for sprite-sheet animation or random index selection.

Syntax

```c
int GetSpriteSheetCellEndIndex()
```

**SetSpriteSheetCellEndIndex**

Sets the end index of the sprite-sheet cell range that is used for sprite-sheet animation or random index selection.

Syntax

```c
void SetSpriteSheetCellEndIndex(int spriteSheetEndIndex)
```
**GetSpriteSheetFrameDelay**

Returns, in seconds, the delay between each sprite-sheet frame.

**Syntax**

```
float GetSpriteSheetFrameDelay()
```

**SetSpriteSheetFrameDelay**

Sets, in seconds, the delay between each sprite-sheet frame.

**Syntax**

```
void SetSpriteSheetFrameDelay(float spriteSheetFrameDelay)
```

**GetIsParticleAspectRatioLocked**

Returns whether the width and height of the emitted particles are locked into the current aspect ratio.

**Syntax**

```
bool GetIsParticleAspectRatioLocked()
```

**SetIsParticleAspectRatioLocked**

Sets whether the width and height of the emitted particles are locked into the current aspect ratio.

**Syntax**

```
void SetIsParticleAspectRatioLocked(bool aspectRatioLocked)
```

**GetParticlePivot**

Returns the pivot for the particles in a range from (0,0) at the top left to (1,1) at the bottom right.

**Syntax**

```
AZ::Vector2 GetParticlePivot()
```

**SetParticlePivot**

Sets the pivot for the particles in a range from (0,0) at the top left to (1,1) at the bottom right.

**Syntax**

```
void SetParticlePivot(AZ::Vector2 particlePivot)
```

**GetParticleSize**

Returns the size of the emitted particles.

**Syntax**
AZ::Vector2 GetParticleSize()

**SetParticleSize**

Sets the size of the emitted particles.

**Syntax**

```cpp
void SetParticleSize(AZ::Vector2 particleSize)
```

**GetParticleWidth**

Returns the width of the emitted particles.

**Syntax**

```cpp
float GetParticleWidth()
```

**SetParticleWidth**

Sets the width of the emitted particles.

**Syntax**

```cpp
void SetParticleWidth(float particleWidth)
```

**GetParticleWidthVariation**

Returns the variation in width of the emitted particles.

**Syntax**

```cpp
float GetParticleWidthVariation()
```

**SetParticleWidthVariation**

Sets the variation in width of the emitted particles.

**Syntax**

```cpp
void SetParticleWidthVariation(float particleWidthVariation)
```

**GetParticleHeight**

Returns the height of the emitted particles.

**Syntax**

```cpp
float GetParticleHeight()
```

**SetParticleHeight**

Sets the height of the emitted particles.
**void SetParticleHeight(float particleHeight)**

### GetParticleHeightVariation

Returns the variation in height of the emitted particles.

**Syntax**

```c
float GetParticleHeightVariation()
```

### SetParticleHeightVariation

Sets the variation in height of the emitted particles.

**Syntax**

```c
void SetParticleHeightVariation(float particleHeightVariation)
```

### GetParticleMovementCoordinateType

Returns the co-ordinate system used for the movement of the emitted particles.

**Syntax**

```c
eUiParticleCoordinateType GetParticleMovementCoordinateType()
```

Possible values for the movement space are as follows.

```c
enum eUiParticleCoordinateType
{
    eUiParticleCoordinateType_Cartesian,
    eUiParticleCoordinateType_Polar
};
```

### SetParticleMovementCoordinateType

Sets the coordinate system that is used for the movement of the emitted particles.

**Syntax**

```c
void SetParticleMovementCoordinateType(eUiParticleCoordinateType movementCoordinateType)
```

For possible values for the coordinate type, see **GetParticleMovementCoordinateType (p. 3101)**.

### GetParticleAccelerationMovementSpace

Returns the coordinate system that is used for the acceleration of particles.

**Syntax**

```c
eUiParticleCoordinateType GetParticleAccelerationMovementSpace()
```

For possible values for the movement space, see **GetParticleMovementCoordinateType (p. 3101)**.
**SetParticleAccelerationMovementSpace**

Sets the coordinate system that is used for the acceleration of particles.

**Syntax**

```cpp
void SetParticleAccelerationMovementSpace(eUiParticleCoordinateType accelerationMovementSpace)
```

For possible values for the movement space, see `GetParticleMovementCoordinateType` (p. 3101).

**GetParticleInitialVelocity**

Returns the initial velocity of the emitted particles. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```cpp
AZ::Vector2 GetParticleInitialVelocity()
```

**SetParticleInitialVelocity**

Sets the initial velocity of the emitted particles. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```cpp
void SetParticleInitialVelocity(AZ::Vector2 particleInitialVelocity)
```

**GetParticleInitialVelocityVariation**

Returns the variation in the particle initial velocity. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```cpp
AZ::Vector2 GetParticleInitialVelocityVariation()
```

**SetParticleInitialVelocityVariation**

Sets the variation in the particle initial velocity. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```cpp
void SetParticleInitialVelocityVariation(AZ::Vector2 initialVelocityVariation)
```

**GetParticleSpeed**

Returns the initial particle speed. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```cpp
float GetParticleSpeed()
```
**SetParticleSpeed**

Sets the initial particle speed. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```c
void SetParticleSpeed(float particleSpeed)
```

**GetParticleSpeedVariation**

Returns the variation in initial particle speed. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```c
float GetParticleSpeedVariation()
```

**SetParticleSpeedVariation**

Sets the variation in initial particle speed. Applicable only when the emitter doesn't control the emit direction.

**Syntax**

```c
void SetParticleSpeedVariation(float particleSpeedVariation)
```

**GetParticleAcceleration**

Returns the acceleration of the emitted particles.

**Syntax**

```c
AZ::Vector2 GetParticleAcceleration()
```

**SetParticleAcceleration**

Sets the acceleration of the emitted particles.

**Syntax**

```c
void SetParticleAcceleration(AZ::Vector2 particleAcceleration)
```

**GetIsParticleRotationFromVelocity**

Returns whether the particle is oriented so that its top points towards the current velocity vector.

**Syntax**

```c
bool GetIsParticleRotationFromVelocity()
```

**SetIsParticleRotationFromVelocity**

Sets whether the particle is oriented so that its top points towards the current velocity vector.

**Syntax**
void SetIsParticleRotationFromVelocity(bool rotationFromVelocity)

GetIsParticleInitialRotationFromInitialVelocity
Returns whether the particle is initially oriented so that its top points towards the initial velocity vector.

Syntax
bool GetIsParticleInitialRotationFromInitialVelocity()

SetIsParticleInitialRotationFromInitialVelocity
Sets whether the particle is initially oriented so that its top points towards the initial velocity vector.

Syntax
void SetIsParticleInitialRotationFromInitialVelocity(bool initialRotationFromVelocity)

GetParticleInitialRotation
Returns, in degrees clockwise measured from straight up, the initial rotation.

Syntax
float GetParticleInitialRotation()

SetParticleInitialRotation
Sets, in degrees clockwise measured from straight up, the initial rotation.

Syntax
void SetParticleInitialRotation(float particleInitialRotation)

GetParticleInitialRotationVariation
Returns, in degrees clockwise measured from straight up, the variation of the initial rotation.

Syntax
float GetParticleInitialRotationVariation()

SetParticleInitialRotationVariation
Sets, in degrees clockwise measured from straight up, the variation of the initial rotation.

Syntax
void SetParticleInitialRotation(float particleInitialRotationVariation)

GetParticleRotationSpeed
Returns, in degrees clockwise per second, the rotation speed of the emitted particles.
Syntax

```csharp
float GetParticleRotationSpeed()
```

**SetParticleRotationSpeed**

Sets, in degrees clockwise per second, the rotation speed of the emitted particles.

Syntax

```csharp
void SetParticleRotationSpeed(float rotationSpeed)
```

**GetParticleRotationSpeedVariation**

Returns, in degrees clockwise per second, the variation in rotation speed of the emitted particles.

Syntax

```csharp
float GetParticleRotationSpeedVariation()
```

**SetParticleRotationSpeedVariation**

Sets, in degrees clockwise per second, the variation in rotation speed of the emitted particles.

Syntax

```csharp
void SetParticleRotationSpeedVariation(float rotationSpeedVariation)
```

**GetParticleColor**

Returns the color of the emitted particles.

Syntax

```csharp
AZ::Color GetParticleColor()
```

**SetParticleColor**

Sets the color of the emitted particles.

Syntax

```csharp
void SetParticleColor(AZ::Color particleColor)
```

**GetParticleColorBrightnessVariation**

Returns, in the range [0,1], the variation in color brightness of the emitted particles.

Syntax

```csharp
float GetParticleColorBrightnessVariation()
```

**SetParticleColorBrightnessVariation**

Sets, in the range [0,1], the variation in color brightness of the emitted particles.
### Syntax

```c
void SetParticleColorBrightnessVariation(float brightnessVariation)
```

**GetParticleColorTintVariation**

Returns, in the range [0,1], the variation in color tint of the emitted particles.

### Syntax

```c
float GetParticleColorTintVariation()
```

**SetParticleColorTintVariation**

Sets, in the range [0,1], the variation in color tint of the emitted particles.

### Syntax

```c
void SetParticleColorTintVariation(float tintVariation)
```

**GetParticleAlpha**

Returns, in the range [0,1], the alpha of the emitted particles.

### Syntax

```c
float GetParticleAlpha()
```

**SetParticleAlpha**

Sets, in the range [0,1], the alpha of the emitted particles.

### Syntax

```c
void SetParticleAlpha(float particleAlpha)
```

### UiRadioButtonComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use **UiRadioButtonComponent** to implement a UI radio button in Amazon Lumberyard.

### UiRadioButtonBus

Services messages for the **UiRadioButtonComponent**.

**GetState**

Returns the state of the radio button. True if selected; false otherwise.

### Syntax
bool GetState()

**GetGroup**

Returns the group of the radio button.

**Syntax**

```cpp
AZ::EntityId GetGroup()
```

**GetCheckedEntity**

Returns the child element that is shown when the radio button is selected.

**Syntax**

```cpp
AZ::EntityId GetCheckedEntity()
```

**SetCheckedEntity**

Sets the child element to show when the radio button is selected.

**Syntax**

```cpp
void SetCheckedEntity(AZ::EntityId entityId)
```

**GetUncheckedEntity**

Returns the child element that is shown when the radio button is cleared.

**Syntax**

```cpp
AZ::EntityId GetUncheckedEntity()
```

**SetUncheckedEntity**

Sets the child element to show when the radio button is cleared.

**Syntax**

```cpp
void SetUncheckedEntity(AZ::EntityId entityId)
```

**GetTurnOnActionName**

Returns the name of the action that is triggered when the radio button is selected.

**Syntax**

```cpp
const AZStd::string& GetTurnOnActionName()
```

**SetTurnOnActionName**

Sets the action triggered when the radio button is selected.
UiRadioButtonComponent

Syntax

void SetTurnOnActionName(const AZStd::string & actionName)

GetTurnOffActionName

Returns the action triggered when the radio button is cleared.

Syntax

const AZStd::string & GetTurnOffActionName()

SetTurnOffActionName

Sets the action triggered when the radio button is cleared.

Syntax

void SetTurnOffActionName(const AZStd::string & actionName)

GetChangedActionName

Returns the action triggered when the radio button value changes.

Syntax

AZStd::string & GetChangedActionName()

SetChangedActionName

Sets the action triggered when the radio button value changes.

Syntax

void SetChangedActionName(const AZStd::string & actionName)

UiRadioButtonCommunicationBus

Allows communication between the radio button group and a radio button.

SetState

Sets the state of the radio button. True if selected; false otherwise.

Syntax

void SetState(bool checked)

SetGroup

Sets the group of the radio button.

Syntax
void SetGroup(AZ::EntityId group)

**UiRadioButtonNotificationBus**

Services notifications for the UiRadioButtonComponent.

**OnRadioButtonStateChanged**

Notifies that the radio button state has changed.

**Syntax**

```cpp
void OnRadioButtonStateChanged(bool checked)
```

**UIScrollBarComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.

[Download O3DE](https://aws.amazon.com) or visit the [AWS Game Tech blog](https://aws.amazon.com) to learn more.

Controls scroll bar characteristics.

**UIScrollBarBus**

Services messages for the UiScrollBarComponent.

**GetAutoFadeDelay**

Returns the fade delay in seconds.

**Minimum Lumberyard Version:** 1.24

**Syntax**

```cpp
float GetAutoFadeDelay()
```

**GetAutoFadeSpeed**

Returns the fade speed in seconds.

**Minimum Lumberyard Version:** 1.24

**Syntax**

```cpp
float GetAutoFadeSpeed()
```

**GetHandleEntity**

Returns the handle entity.

**Syntax**

```cpp
AZ::EntityId GetHandleEntity()
```
GetHandleSize
Returns the size of the handle relative to the scroll bar (0 - 1).

Syntax
float GetHandleSize()

GetMinHandlePixelSize
Returns the minimum size of the handle in pixels.

Syntax
float GetMinHandlePixelSize()

IsAutoFadeEnabled
Returns whether auto fade is enabled.

Minimum Lumberyard Version: 1.24

Syntax
bool IsAutoFadeEnabled()

SetAutoFadeDelay
Sets the fade delay in seconds.

Minimum Lumberyard Version: 1.24

Syntax
void SetAutoFadeDelay(float delay)

SetAutoFadeEnabled
Sets whether auto fade is enabled.

Minimum Lumberyard Version: 1.24

Syntax
void SetAutoFadeEnabled(bool isEnabled)

SetAutoFadeSpeed
Sets the fade speed in seconds.

Minimum Lumberyard Version: 1.24

Syntax
void SetAutoFadeSpeed(float speed)
SetHandleEntity
Sets the handle entity.

Syntax

```c
void SetHandleEntity(AZ::EntityId entityId)
```

SetHandleSize
Sets the size of the handle relative to the scroll bar (0 - 1).

Syntax

```c
void SetHandleSize(float size)
```

SetMinHandlePixelSize
Sets the minimum size of the handle in pixels.

Syntax

```c
void SetMinHandlePixelSize(float size)
```

UiScrollerBus
Services scrolling for the UIScrollBarComponent.

GetOrientation
Returns the orientation of the scroller.

Syntax

```c
eUiScrollerOrientation GetOrientation()
```

Following are possible values for eUiScrollerOrientation.

```c
enum eUiScrollerOrientation
{
  eUiScrollerOrientation_Horizontal,
  eUiScrollerOrientation_Vertical
};
```

GetValue
Returns the current value for the scroller (0 - 1).

Syntax

```c
float GetValue()
```

GetValueChangedActionName
Returns the name of the action triggered when the value has changed.
GetValueChangingActionName
Returns the name of the action triggered while the value is changing.

Syntax

```cpp
const AZStd::string& GetValueChangingActionName()
```

SetOrientation
Sets the orientation of the scroller.

Syntax

```cpp
void SetOrientation(eUiScrollerOrientation orientation)
```

For possible values for `eUiScrollerOrientation`, see `GetOrientation (p. 3111)`.

SetValue
Sets the value of the scroller (0 - 1).

Syntax

```cpp
void SetValue(float value)
```

SetValueChangedActionName
Sets the action triggered when the value has changed.

Syntax

```cpp
void SetValueChangedActionName(const AZStd::string& actionName)
```

SetValueChangingActionName
Sets the name of the action triggered while the value is changing.

Syntax

```cpp
void SetValueChangingActionName(const AZStd::string& actionName)
```

UIScrollBoxComponent
Controls the characteristics of a scroll box.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
UiScrollBoxBus

Services messages for the UiScrollBoxComponent.

FindClosestContentChildElement

Finds the child of the content element that is closest to the content anchors at the current scroll offset (the currently selected child).

Syntax

```
AZ::EntityId FindClosestContentChildElement()
```

GetContentEntity

Returns the content element for the scroll box.

Syntax

```
AZ::EntityId GetContentEntity()
```

GetHorizontalScrollBarEntity

Returns the horizontal scroll bar element for the scroll box.

Syntax

```
AZ::EntityId GetHorizontalScrollBarEntity()
```

GetHorizontalScrollBarVisibility

Returns the visibility behavior for the horizontal scroll bar of the scroll box.

Syntax

```
eUiScrollBoxScrollBarVisibility GetHorizontalScrollBarVisibility()
```

Following are possible values for eUiScrollBoxScrollBarVisibility.

```
enum eUiScrollBoxScrollBarVisibility
{
    eUiScrollBoxScrollBarVisibility_AlwaysShow,
    eUiScrollBoxScrollBarVisibility_AutoHide,
    eUiScrollBoxScrollBarVisibility_AutoHideAndResizeViewport
};
```

GetIsHorizontalScrollingEnabled

Returns whether the scroll box allows horizontal scrolling.

Syntax

```
bool GetIsHorizontalScrollingEnabled()
```
GetIsScrollingConstrained
Returns whether the scroll box restricts scrolling to the content area.

Syntax

```cpp
bool GetIsScrollingConstrained()
```

GetIsVerticalScrollingEnabled
Returns whether the scroll box allows vertical scrolling.

Syntax

```cpp
bool GetIsVerticalScrollingEnabled()
```

GetNormalizedScrollValue
Returns the scroll value from 0 – 1.

Syntax

```cpp
AZ::Vector2 GetNormalizedScrollValue()
```

GetScrollOffset
Returns the scroll offset of the scroll box. The scroll offset is the offset from the content element's anchor point to the content element's pivot.

Syntax

```cpp
AZ::Vector2 GetScrollOffset()
```

GetScrollOffsetChangedActionName
Returns the action triggered when the scroll box drag is completed.

Syntax

```cpp
const AZStd::string& GetScrollOffsetChangedActionName()
```

GetScrollOffsetChangingActionName
Returns the action triggered while the scroll box is being dragged.

Syntax

```cpp
AZStd::string& GetScrollOffsetChangingActionName()
```

GetSnapGrid
Returns the snapping grid of the scroll box.

Syntax
GetSnapMode

Returns the snap mode for the scroll box.

Syntax

```cpp
AZ::Vector2 GetSnapGrid()
```

Following are possible values for `eUiScrollBoxSnapMode`.

```cpp
eUiScrollBoxSnapMode
{
    eUiScrollBoxSnapMode_None,
    eUiScrollBoxSnapMode_Children,
    eUiScrollBoxSnapMode_Grid
};
```

GetVerticalScrollBarEntity

Returns the vertical scroll bar element for the scroll box.

Syntax

```cpp
AZ::EntityId GetVerticalScrollBarEntity()
```

GetVerticalScrollBarVisibility

Returns the visibility behavior for the vertical scroll bar of the scroll box.

Syntax

```cpp
eUiScrollBoxScrollBarVisibility GetVerticalScrollBarVisibility()
```

HasHorizontalContentToScroll

Returns whether there is content to scroll horizontally.

Syntax

```cpp
bool HasHorizontalContentToScroll()
```

HasVerticalContentToScroll

Returns whether there is content to scroll vertically.

Syntax

```cpp
bool HasVerticalContentToScroll()
```

SetContentEntity

Sets the content element for the scroll box.
**Syntax**

```cpp
void SetContentEntity(AZ::EntityId entityId)
```

**SetHorizontalScrollBarEntity**

Sets the horizontal scroll bar element for the scroll box.

**Syntax**

```cpp
void SetHorizontalScrollBarEntity(AZ::EntityId entityId)
```

**SetHorizontalScrollBarVisibility**

Sets the visibility behavior for the horizontal scroll bar of the scroll box.

**Syntax**

```cpp
void SetHorizontalScrollBarVisibility(eUiScrollBoxScrollBarVisibility visibility)
```

For possible values for `eUiScrollBoxScrollBarVisibility`, see `GetHorizontalScrollBarVisibility (p. 3113)`.

**SetIsHorizontalScrollingEnabled**

Sets whether the scroll box allows horizontal scrolling.

**Syntax**

```cpp
void setIsHorizontalScrollingEnabled(bool isEnabled)
```

**SetIsScrollingConstrained**

Sets whether the scroll box restricts scrolling to the content area.

**Syntax**

```cpp
void SetIsScrollingConstrained(bool isConstrained)
```

**SetIsVerticalScrollingEnabled**

Sets whether the scroll box allows vertical scrolling.

**Syntax**

```cpp
void setIsVerticalScrollingEnabled(bool isEnabled)
```

**SetScrollOffset**

Sets the scroll offset of the scroll box.

**Syntax**

```cpp
void SetScrollOffset(AZ::Vector2 scrollOffset)
```
**SetScrollOffsetChangedActionName**

Sets the action triggered when the scroll box drag is completed.

**Syntax**

```c
void SetScrollOffsetChangedActionName(const AZStd::string& actionName)
```

**SetScrollOffsetChangingActionName**

Sets the action triggered while the scroll box is being dragged.

**Syntax**

```c
void SetScrollOffsetChangingActionName(const AZStd::string& actionName)
```

**SetSnapGrid**

Sets the snapping grid of the scroll box.

**Syntax**

```c
void SetSnapGrid(AZ::Vector2 snapGrid)
```

**SetSnapMode**

Sets the snap mode for the scroll box.

**Syntax**

```c
void SetSnapMode(eUiScrollBoxSnapMode snapMode)
```

For possible values for `eUiScrollBoxSnapMode`, see [GetSnapMode](p. 3115).

**SetVerticalScrollBarEntity**

Sets the vertical scroll bar element for the scroll box.

**Syntax**

```c
void SetVerticalScrollBarEntity(AZ::EntityId entityId)
```

**SetVerticalScrollBarVisibility**

Sets the visibility behavior for the vertical scroll bar of the scroll box.

**Syntax**

```c
void SetVerticalScrollBarVisibility(eUiScrollBoxScrollBarVisibility visibility)
```

**UiScrollBoxNotificationBus**

Services scroll offset change notifications for the `UiScrollBoxComponent`. 
**OnScrollOffsetChanged**
Called when the scroll offset has changed.

**Syntax**

```csharp
void OnScrollOffsetChanged(AZ::Vector2 newScrollOffset)
```

**OnScrollOffsetChanging**
Called when the scroll offset is changing.

**Syntax**

```csharp
void OnScrollOffsetChanging(AZ::Vector2 newScrollOffset)
```

**UiScrollableNotificationBus**
Services scrollable value change notifications for the **UiScrollBoxComponent**.

**OnScrollableValueChanged**
Called when the scroll value (0 - 1) has changed.

**Syntax**

```csharp
void OnScrollableValueChanged(AZ::Vector2 value)
```

**OnScrollableValueChanging**
Called when the scroll value (0 - 1) is changing.

**Syntax**

```csharp
void OnScrollableValueChanging(AZ::Vector2 value)
```

**UISliderComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls the values of a slider.

**UISliderBus**
Services messages for the **UISliderComponent**.

**GetFillEntity**
Returns the fill element.

**Syntax**
**GetManipulatorEntity**

Returns the entity ID of the manipulator element.

**Syntax**

```cpp
AZ::EntityId GetManipulatorEntity()
```

**GetMaxValue**

Returns the maximum value of the slider.

**Syntax**

```cpp
float GetMaxValue()
```

**GetMinValue**

Returns the minimum value of the slider.

**Syntax**

```cpp
float GetMinValue()
```

**GetStepValue**

Returns the smallest increment allowed between values. Zero means no restriction.

**Syntax**

```cpp
float GetStepValue()
```

**GetTrackEntity**

Returns the track element.

**Syntax**

```cpp
AZ::EntityId GetTrackEntity()
```

**GetValue**

Returns the value of the slider.

**Syntax**

```cpp
float GetValue()
```

**GetValueChangedActionName**

Returns the action triggered when the value has finished changing.
**Syntax**

```cpp
const AZStd::string& GetValueChangedActionName()
```

**GetValueChangingActionName**

Returns the name of the action triggered while the value is changing.

**Syntax**

```cpp
const AZStd::string& GetValueChangingActionName()
```

**SetFillEntity**

Sets the fill element.

**Syntax**

```cpp
void SetFillEntity(AZ::EntityId entityId)
```

**SetManipulatorEntity**

Sets the manipulator element.

**Syntax**

```cpp
void SetManipulatorEntity(AZ::EntityId entityId)
```

**SetMaxValue**

Sets the maximum value of the slider.

**Syntax**

```cpp
void SetMaxValue(float value)
```

**SetMinValue**

Sets the minimum value of the slider.

**Syntax**

```cpp
void SetMinValue(float value)
```

**SetStepValue**

Sets the smallest increment allowed between values. Use zero for no restriction.

**Syntax**

```cpp
void SetStepValue(float step)
```

**SetTrackEntity**

Sets the track element.
Syntax

```cpp
void SetTrackEntity(AZ::EntityId entityId)
```

**SetValue**

Sets the value of the slider.

**Syntax**

```cpp
void SetValue(float value)
```

**SetValueChangedActionName**

Sets the action triggered when the value is done changing.

**Syntax**

```cpp
void SetValueChangedActionName(const AZStd::string& actionName)
```

**SetValueChangingActionName**

Sets the action triggered while the value is changing.

**Syntax**

```cpp
void SetValueChangingActionName(const AZStd::string& actionName)
```

**UiSliderNotificationBus**

Services notifications for the `UiSliderComponent`.

**OnSliderValueChanged**

The slider value has finished changing.

**Syntax**

```cpp
void OnSliderValueChanged(float value)
```

**OnSliderValueChanging**

The slider value is changing.

**Syntax**

```cpp
void OnSliderValueChanging(float value)
```

**UITextComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Controls the text and formatting of a text element.

**UiTextBus**

Services messages for the UiTextComponent.

**GetColor**

Returns the color to draw the text string.

**Syntax**

```cpp
AZ::Color GetColor()
```

**GetFont**

Returns the pathname to the font.

**Syntax**

```cpp
AZStd::string GetFont()
```

**GetFontEffect**

Returns the font effect.

**Syntax**

```cpp
int GetFontEffect()
```

**GetFontSize**

Returns the size of the font in points.

**Syntax**

```cpp
float GetFontSize()
```

**GetHorizontalAlignment**

Returns the horizontal text alignment.

**Syntax**

```cpp
eUiHAlign GetHorizontalAlignment()
```

Following are possible values for eUiHAlign.

```cpp
enum eUiHAlign
{
   eUiHAlign_Left,
   eUiHAlign_Center,
   eUiHAlign_Right
};
```
**GetOverflowMode**

Returns the overflow behavior of the text.

**Syntax**

```cpp
eUiTextOverflowMode GetOverflowMode()
```

Following are possible values for `eUiTextOverflowMode`.

```cpp
enum eUiTextOverflowMode
{
    eUiTextOverflowMode_OverflowText,
    eUiTextOverflowMode_ClipText
};
```

**GetText**

Returns the text string being displayed by the element.

**Syntax**

```cpp
AZStd::string GetText()
```

**GetVerticalTextAlignment**

Returns the vertical text alignment.

**Syntax**

```cpp
eUiVAlign GetVerticalTextAlignment()
```

Following are possible values for `eUiVAlign`.

```cpp
enum eUiVAlign
{
    eUiVAlign_Top,
    eUiVAlign_Center,
    eUiVAlign_Bottom
};
```

**GetWrapText**

Returns whether text is wrapped.

**Syntax**

```cpp
eUiTextWrapTextSetting GetWrapText()
```

Following are possible values for `eUiTextWrapTextSetting`.

```cpp
enum eUiTextWrapTextSetting
{
    eUiTextWrapTextSetting_NoWrap,
    eUiTextWrapTextSetting_Wrap
};
```
**SetColor**
Sets the color to draw the text string.

**Syntax**
```cpp
void SetColor(const AZ::Color& color)
```

**SetFont**
Sets the pathname to the font.

**Syntax**
```cpp
void SetFont(const AZStd::string& fontPath)
```

**SetFontEffect**
Sets the font effect.

**Syntax**
```cpp
void SetFontEffect(int effectIndex)
```

**SetFontSize**
Sets the size of the font in points.

**Syntax**
```cpp
void SetFontSize(float size)
```

**SetHorizontalTextAlignment**
Sets the horizontal text alignment.

**Syntax**
```cpp
void SetHorizontalTextAlignment(eUiHAlign alignment)
```

For possible values for `eUiHAlign`, see `GetHorizontalTextAlignment (p. 3122)`.

**SetOverflowMode**
Sets the overflow behavior of the text.

**Syntax**
```cpp
void SetOverflowMode(eUiTextOverflowMode overflowMode)
```

For possible values for `eUiTextOverflowMode`, see `GetOverflowMode (p. 3123)`.

**SetText**
Sets the text string being displayed by the element.
You can use a text input component to provide player text input capability.

**UiTextInputBus**

Services messages for the UiTextInputComponent.

**GetChangeAction**

Returns the action triggered when the text is changed.

**Syntax**

```cpp
const AZStd::string& GetChangeAction()
```

**GetCursorBlinkInterval**

Returns the cursor blink interval of the text input.

**Syntax**

```cpp
float GetCursorBlinkInterval()
```
**GetEndEditAction**

Returns the action triggered when the editing of text is finished.

**Syntax**

```cpp
const AZStd::string& GetEndEditAction()
```

**GetEnterAction**

Returns the action triggered when enter is pressed.

**Syntax**

```cpp
const AZStd::string& GetEnterAction()
```

**GetIsPasswordField**

Returns whether the text input is configured as a password field.

**Syntax**

```cpp
bool GetIsPasswordField()
```

**GetMaxStringLength**

Returns the maximum number of characters that can be entered.

**Syntax**

```cpp
int GetMaxStringLength()
```

**GetPlaceHolderTextEntity**

Returns the placeholder text element.

**Syntax**

```cpp
AZ::EntityId GetPlaceHolderTextEntity()
```

**GetReplacementCharacter**

Returns the replacement character used to hide password text.

**Syntax**

```cpp
char GetReplacementCharacter()
```

**GetText**

Returns the text string being displayed or edited by the element.

**Syntax**

```cpp
AZStd::string GetText()
```
**GetTextCursorColor**
Returns the color to be used for the text cursor.

**Syntax**

```cpp
AZ::Color GetTextCursorColor()
```

**GetTextEntity**
Returns the text element.

**Syntax**

```cpp
AZ::EntityId GetTextEntity()
```

**GetTextSelectionColor**
Returns the color to be used for the text background when it is selected.

**Syntax**

```cpp
AZ::Color GetTextSelectionColor()
```

**SetChangeAction**
Sets the action triggered when the text is changed.

**Syntax**

```cpp
void SetChangeAction(const AZStd::string& actionName)
```

**SetCursorBlinkInterval**
Sets the cursor blink interval of the text input.

**Syntax**

```cpp
void SetCursorBlinkInterval(float interval)
```

**SetEndEditAction**
Sets the action triggered when the editing of text is finished.

**Syntax**

```cpp
void SetEndEditAction(const AZStd::string& actionName)
```

**SetEnterAction**
Sets the action triggered when enter is pressed.

**Syntax**

```cpp
void SetEnterAction(const AZStd::string& actionName)
```
**SetIsPasswordField**
Sets whether the text input is configured as a password field.

**Syntax**

```cpp
void SetIsPasswordField(bool passwordField)
```

**SetMaxStringLength**
Sets the maximum number of characters that can be entered.

**Syntax**

```cpp
void SetMaxStringLength(int maxCharacters)
```

**SetPlaceHolderTextEntity**
Sets the placeholder text element.

**Syntax**

```cpp
void SetPlaceHolderTextEntity(AZ::EntityId textEntity)
```

**SetReplacementCharacter**
Sets the replacement character used to hide password text.

**Syntax**

```cpp
void SetReplacementCharacter(char replacementChar)
```

**SetText**
Sets the text string being displayed or edited by the element.

**Syntax**

```cpp
void SetText(const AZStd::string& text)
```

**SetTextCursorColor**
Sets the color to be used for the text cursor.

**Syntax**

```cpp
void SetTextCursorColor(const AZ::Color& color)
```

**SetTextEntity**
Sets the text element.

**Syntax**

```cpp
void SetTextEntity
```
void SetTextEntity(AZ::EntityId textEntity)

SetTextSelectionColor

Sets the color to be used for the text background when it is selected.

Syntax

void SetTextSelectionColor(const AZ::Color& color)

UiTextInputNotificationBus

Services notifications for the UiTextInputComponent.

OnTextInputChange

Called when a character is added, removed, or changed.

Syntax

void OnTextInputChange(const AZStd::string& textString)

OnTextInputEndEdit

Called when edit of text is completed.

Syntax

void OnTextInputEndEdit(const AZStd::string& textString)

OnTextInputEnter

Called when Enter is pressed on the keyboard.

Syntax

void OnTextInputEnter(const AZStd::string& textString)

UiTooltipComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use a tooltip component to provide the text of a tooltip.

UiTooltipBus

Services messages for the UiTooltipComponent.

GetText

Returns the tooltip text.
### Syntax

```
AZStd::string GetText()
```

### SetText

Sets the tooltip text.

### Syntax

```
void SetText(const AZStd::string& text)
```

## UITooltipDisplayComponent

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls the display behavior of a tooltip.

## UiTooltipDisplayBus

Services messages for the UiTooltipDisplayComponent.

### GetAutoPosition

Returns whether the tooltip display element is auto positioned.

### Syntax

```
bool GetAutoPosition()
```

### GetAutoPositionMode

Returns the auto position mode.

### Syntax

```
AutoPositionMode GetAutoPositionMode()
```

Following are possible values for `AutoPositionMode`.

```
enum AutoPositionMode
{
    OffsetFromMouse,
    OffsetFromElement
};
```

### GetAutoSize

Returns whether the tooltip display element should be resized so that the text element size matches the size of the string.

### Syntax
bool GetAutoSize()

GetDelayTime
Returns the amount of time to wait before showing the tooltip display element after the trigger condition has occurred.

Syntax
float GetDelayTime()

GetDisplayTime
Returns the amount of time the tooltip display element is to remain visible.

Syntax
float GetDisplayTime()

GetOffset
Returns the offset from the tooltip display element's pivot to the mouse position.

Syntax
const AZ::Vector2& GetOffset()

GetTextEntity
Returns the entity ID of the text element that is used for resizing.

Syntax
AZ::EntityId GetTextEntity()

GetTriggerMode
Returns the trigger mode describing the condition under which the tooltip will be displayed.

Minimum Lumberyard Version: 1.24

Syntax
TriggerMode GetTriggerMode()

Following are possible values for TriggerMode.

enum TriggerMode
{
    OnHover,
    OnPress,
    OnClick
};
**SetAutoPosition**

Sets whether the tooltip display element is auto positioned.

**Syntax**

```csharp
void SetAutoPosition(bool autoPosition)
```

**SetAutoPositionMode**

Sets the auto position mode.

**Syntax**

```csharp
void SetAutoPositionMode(AutoPositionMode autoPositionMode)
```

Following are possible values for `AutoPositionMode`.

```csharp
enum AutoPositionMode
{
    OffsetFromMouse,
    OffsetFromElement
};
```

**SetAutoSize**

Sets whether the tooltip display element should be resized so that the text element size matches the size of the string.

**Syntax**

```csharp
void SetAutoSize(bool autoSize)
```

**SetDelayTime**

Sets the amount of time to wait before showing the tooltip display element after the trigger condition has occurred.

**Syntax**

```csharp
void SetDelayTime(float delayTime)
```

**SetDisplayTime**

Sets the amount of time the tooltip display element is to remain visible.

**Syntax**

```csharp
void SetDisplayTime(float displayTime)
```

**SetOffset**

Sets the offset from the tooltip display element's pivot to the mouse position.

**Syntax**
void SetOffset(const AZ::Vector2& offset)

**SetTextEntity**

Sets the entity ID of the text element that is used for resizing. The text element must be a child of this entity.

**Syntax**

void SetTextEntity(AZ::EntityId textEntity)

**SetTriggerMode**

Sets the trigger condition for displaying the tooltip after the set amount of delay time has elapsed.

**Minimum Lumberyard Version:** 1.24

**Syntax**

void SetTriggerMode(TriggerMode triggerMode)

Following are possible values for TriggerMode.

```cpp
template
{
    OnHover,
    OnPress,
    OnClick
};
```

**UITransform2dComponent**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls positioning, scaling, rotation, anchor, and offset settings for UI elements.

**UiTransformBus**

Services messages for the UiTransform2dComponent.

**GetCanvasPosition**

Returns the position for this element in the canvas space.

**Syntax**

AZ::Vector2 GetCanvasPosition()

**GetLocalPosition**

Returns the position for this element relative to the center of the element's anchors.
UITransform2dComponent

Syntax

```
AZ::Vector2 GetLocalPosition()
```

GetPivot

Returns the pivot point.

Syntax

```
AZ::Vector2 GetPivot()
```

GetScale

Returns the scale.

Syntax

```
AZ::Vector2 GetScale()
```

GetScaleToDeviceMode

Returns how this element and all of the child elements are scaled to allow for the difference between the authored canvas size and the actual viewport size.

Syntax

```
ScaleToDeviceMode GetScaleToDeviceMode()
```

Following are possible values for ScaleToDeviceMode.

```
enum ScaleToDeviceMode
{
    None,
    UniformScaleToFit,
    UniformScaleToFill,
    UniformScaleToFitX,
    UniformScaleToFitY,
    NonUniformScale,
    ScaleXOnly,
    ScaleYOnly
};
```

GetViewportPosition

Returns the position for this element in the viewport space.

Syntax

```
AZ::Vector2 GetViewportPosition()
```

GetZRotation

Returns the rotation about the z-axis.

Syntax
```csharp
float GetZRotation()

**MoveCanvasPositionBy**

Moves this element in the canvas space.

**Syntax**

```csharp
void MoveCanvasPositionBy(const AZ::Vector2& offset)
```

**MoveLocalPositionBy**

Moves this element relative to the center of the element's anchors.

**Syntax**

```csharp
void MoveLocalPositionBy(const AZ::Vector2& offset)
```

**MoveViewportPositionBy**

Moves this element in the viewport space.

**Syntax**

```csharp
void MoveViewportPositionBy(const AZ::Vector2& offset)
```

**SetCanvasPosition**

Sets the position for this element in the canvas space.

**Syntax**

```csharp
void SetCanvasPosition(const AZ::Vector2& position)
```

**SetLocalPosition**

Sets the position for this element relative to the center of the element's anchors.

**Syntax**

```csharp
void SetLocalPosition(const AZ::Vector2& position)
```

**SetPivot**

Sets the pivot point.

**Syntax**

```csharp
void SetPivot(AZ::Vector2 pivot)
```

**SetScale**

Sets the scale.
Syntax

```cpp
void SetScale(AZ::Vector2 scale)
```

**SetScaleToDeviceMode**

Sets how the element and all of its child elements are scaled to allow for the difference between the authored canvas size and the actual viewport size.

**Syntax**

```cpp
void SetScaleToDeviceMode(ScaleToDeviceMode scaleToDeviceMode)
```

Following are possible values for `scaleToDeviceMode`.

```cpp
enum ScaleToDeviceMode {
    None,
    UniformScaleToFit,
    UniformScaleToFill,
    UniformScaleToFitX,
    UniformScaleToFitY,
    NonUniformScale,
    ScaleXOnly,
    ScaleYOnly
};
```

**SetViewportPosition**

Sets the position for this element in the viewport space.

**Syntax**

```cpp
void SetViewportPosition(const AZ::Vector2& position)
```

**SetZRotation**

Sets the rotation about the z-axis.

**Syntax**

```cpp
void SetZRotation(float rotation)
```

**UiTransform2dBus**

Services anchors and offsets for the UiTransform2dComponent.

**GetAnchors**

Returns the UI anchors.

**Syntax**

```cpp
UiAnchors GetAnchors()
```
Following are possible values for `UiAnchors`.

```cpp
class UiAnchors {
    float left;
    float top;
    float right;
    float bottom;
};
```

**GetOffsets**

Returns the UI offsets.

**Syntax**

```cpp
UiOffsets GetOffsets()
```

The following are possible values for `UiOffsets`.

```cpp
class UiOffsets {
    float left;
    float top;
    float right;
    float bottom;
};
```

**SetAnchors**

Sets the anchors.

**Syntax**

```cpp
void SetAnchors(UiAnchors anchors, bool adjustOffsets, bool allowPush)
```

For possible values for `UiAnchors`, see [GetAnchors (p. 3136)](#).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustOffsets</td>
<td>If true, the offsets are adjusted to keep the rectangle in the same position.</td>
</tr>
<tr>
<td>allowPush</td>
<td>Only takes effect if the anchors are invalid. If true, when an anchor is changed to overlap the anchor opposite it, the opposite anchor moves.</td>
</tr>
</tbody>
</table>

**SetOffsets**

Sets UI offsets.

**Syntax**

```cpp
void SetOffsets(UiOffsets offsets)
```

For possible values for `UiOffsets`, see [GetOffsets (p. 3137)](#).
SetPivotAndAdjustOffsets
Sets the pivot and adjusts the offsets so that this element stays in the same place.

Syntax

```cpp
void SetPivotAndAdjustOffsets(AZ::Vector2 pivot)
```

UI World Components

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Controls the loading and unloading of UI canvases.

UiCanvasAssetRefBus
Services messages for the UiCanvasAssetRefComponent.

LoadCanvas
Loads the UI canvas using the stored asset reference. Returns the entity ID of the loaded canvas.

Syntax

```cpp
AZ::EntityId LoadCanvas()
```

UnloadCanvas
Unloads the UI canvas using the stored asset reference.

Syntax

```cpp
void UnloadCanvas()
```

UiCanvasAssetRefNotificationBus
Handles events sent by the UiCanvasAssetRefComponent.

OnCanvasLoadedIntoEntity
Called when the canvas asset reference loads a UI canvas.

Syntax

```cpp
void OnCanvasLoadedIntoEntity(AZ::EntityId uiCanvasEntity)
```

UiCanvasProxyRefBus
Services messages for the UiCanvasProxyRefComponent.
SetCanvasRefEntity

Sets the entity that manages the UI canvas for this proxy.

Syntax

```cpp
void SetCanvasRefEntity(AZ::EntityId canvasAssetRefEntity)
```

UiCanvasRefBus

Services messages for the UiCanvasAssetRefComponent and UiCanvasProxyRefComponent.

GetCanvas

Returns the UI canvas associated with this entity

Syntax

```cpp
AZ::EntityId GetCanvas()
```

UiCanvasRefNotificationBus

Provides notifications of when the UI canvas reference changes.

OnCanvasRefChanged

Called when the canvas referenced by a UiCanvasAssetRefComponent has changed. This can happen when LoadCanvas (p. 3138), UnloadCanvas (p. 3138), or SetCanvasRefEntity (p. 3139) is called.

Syntax

```cpp
void OnCanvasRefChanged(AZ::EntityId uiCanvasRefEntity, AZ::EntityId uiCanvasEntity)
```
Adding Audio and Sound Effects

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard uses an audio translation layer (ATL) to interface between Lumberyard and third-party audio middleware, so you can develop and change your audio implementation without affecting the game logic. The game logic interacts with ATL controls, which map to their audio middleware equivalents through the Audio Controls Editor. For example, to play a sound, the game executes an ATL trigger, which is mapped to a 'Play' event in the audio middleware.

Lumberyard supports Audiokinetic Wave Works Interactive Sound Engine (Wwise), an audio pipeline solution with which you can create compelling soundscapes for your game. Lumberyard comes pre-configured with Wwise LTX, a free, compact version of Wwise.

For more information, see Setting up Wwise LTX (p. 3150).

Wwise is an industry standard audio middleware, but the full-featured software is not free. For licensing and pricing details, see http://www.audiokinetic.com/pricing.

Topics
- Audio System Overview (p. 3140)
- Audio Translation Layer (p. 3142)
- Audiokinetic Wwise (p. 3149)
- Audio Components (p. 3162)
- Audio Console Variables (p. 3162)

Audio System Overview

The Lumberyard audio system consists of Gems, components, and content.

Gems

Lumberyard provides two audio Gems:

- AudioSystem
- AudioEngineWwise

AudioEngineWwise is an audio engine implementation provided by Lumberyard for Audiokinetic Wwise. An audio engine implementation translates generic Audio Translation Layer (ATL) state requests into real calls to the audio middleware API. It also implements low-level hooks for file I/O and memory allocation as needed for the audio middleware.

The AudioEngineWwise Gem depends on the AudioSystem Gem. It is recommended that you enable both Gems to enable audio, but the AudioSystem Gem has no dependencies and can be enabled by itself.
This leaves open the possibility of other audio middleware Gems to be developed and used instead of AudioEngineWwise.

Lumberyard audio Gems feature the following modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio System</td>
<td>Part of the Audio System Gem. Contains the audio translation layer (ATL) code and manages the state of the audio system in Lumberyard. Most of this module runs on the audio thread, but it also synchronizes with the main thread.</td>
</tr>
<tr>
<td>Audio System Editor</td>
<td>A Lumberyard Editor plugin, and part of the Audio System Gem. Contains the Audio Controls Editor (ACE) to create and manage ATL controls.</td>
</tr>
<tr>
<td>Audio Engine Wwise</td>
<td>Part of the Audio Engine Wwise Gem. Contains the implementation of AudioSystemImplementation interfaces for Wwise. Contains all Audiokinetic APIs. This is the only module that links with Wwise SDK. Can be configured to use Wwise LTX or the full version of Wwise.</td>
</tr>
<tr>
<td>Audio Engine Wwise Editor</td>
<td>A Lumberyard Editor plugin, and part of the Audio Engine Wwise Gem. This is an additional module that the Audio Controls Editor loads when Lumberyard uses Wwise.</td>
</tr>
</tbody>
</table>

**Components**

Core audio components available in Lumberyard Editor enable you to trigger sound effects, play ambient music, change sound variables using RTPC, apply environmental effects, place listeners to act as virtual microphones, and more. For a complete list, see Audio Components (p. 3162).

**Content**

Lumberyard audio features the following content:

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media</td>
<td>Lumberyard loads soundbanks and loose media at runtime. The audio middleware authoring tools compiles and generates the media files.</td>
</tr>
<tr>
<td>Project</td>
<td>The audio middleware authoring tools use a project to manage source audio files, adjust sounds and settings, and generate runtime ready media. The Audio Controls Editor also uses the project to help map ATL controls to the audio middleware equivalents.</td>
</tr>
<tr>
<td>ATL Libraries</td>
<td>When the audio system maps ATL controls to their audio middleware equivalents, it creates ATL libraries, which are saved as XML files.</td>
</tr>
</tbody>
</table>
Audio Translation Layer

Audio Controls Editor

Using the Audio Controls Editor
A. **ATL Controls Pane (p. 3143)** – Hierarchical view of controls that exist in your project. The displayed icon designates the type of control.

B. **Inspector Pane (p. 3144)** – Properties for the control selected in the ATL Controls pane.

C. **Wwise Controls Pane (p. 3146)** – Controls created in the audio middleware authoring application, such as Wwise or Wwise LTX Authoring Tool.

---

### ATL Controls Pane

The ATL Control pane has the following control types. The associated icon designates the type of control. You can customize the name of the control.

#### Audio Controls

<table>
<thead>
<tr>
<th>Audio Control Type</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
</table>
| Trigger            | ![Icon] | Containers that connects to events in the audio middleware. You can author events to complete actions such as play or stop sounds, mute or unmute buses, and so on.  
To preview a trigger, right-click and choose **Execute Trigger**, or press the spacebar. |
<p>| RTPC               | ![Icon] | Real-Time Parameter Control (RTPC) is a floating-point variable that is updated over time by the game logic. RTPCs connect to parameters in the audio middleware that drive and modulate sound characteristics. |
| Switch             | ![Icon] | A variable that can be in one of several states, called switch states, that the game logic can set. For example, a <strong>SurfaceType</strong> switch can have values of Rock, Sand, or Grass. |</p>
<table>
<thead>
<tr>
<th>Audio Control Type</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td><img src="image" alt="Environment Icon" /></td>
<td>Environments can be set on audio objects, which control the amount of environmental effect, such as reverb and echo.</td>
</tr>
<tr>
<td>Preload</td>
<td><img src="image" alt="Preload Icon" /></td>
<td>Preloads are connected to sound banks, which are audio files that include packaged audio data. This audio data contains both signal content and metadata.</td>
</tr>
</tbody>
</table>

**To display a limited subset of control types**

1. In the **Audio Controls Editor**, click **Filters**.
2. Select or clear control types.

![Filters](image)

**To add a new control**

1. Click **Add**.
2. Select the type of control that you want to add.

![Add](image)

**Inspector Pane**

In the **Inspector** pane, you can edit the properties of the control that you selected in the **ATL Controls** pane. You can modify the control's **Name**, select a **Scope**, and modify the **Connected Controls**.
The following table describes the properties that you can modify in the Inspector pane.

### Inspector Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Name of the control. You can customize the name in the ATL Controls pane.</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Controls can exist for a global or on a per-level scope. A control with a global scope exists as long as the game is running and regardless of whether the control is used in the current level. When a specific level is defined as the scope, the controls exist only when that level is loaded. This setting is useful in low-memory systems because controls are loaded only in levels in which they are needed.</td>
</tr>
<tr>
<td><strong>Auto Load</strong></td>
<td>Available only for preloads. If you select Auto Load, the elements preloaded with this control will be reference counted; only one copy of them is created that is shared between all users.</td>
</tr>
<tr>
<td><strong>Preloaded Soundbanks</strong></td>
<td>Available only for preloads. The soundbanks connected with a preload can be different for different platforms. Different soundbanks can be added to different groups, and then in the Platforms field, you can choose which group to load for each platform that you are targeting.</td>
</tr>
<tr>
<td><strong>Platforms</strong></td>
<td>Available only for preloads. You can specify which group of soundbanks to load for each platform. You can share a group between several platforms.</td>
</tr>
<tr>
<td><strong>Connected Controls</strong></td>
<td>Contains the middleware controls connected to your control.</td>
</tr>
</tbody>
</table>

**Note**

The Auto Load, Preloaded Soundbanks, and Platforms properties appear only when the control is a Preload.
Wwise Controls Pane

The controls in the Wwise Controls pane are middleware-specific.

To filter displayed controls

- In the Audio Controls Editor, for Wwise Controls, enter your search term into the Search bar.

To hide controls that are already assigned

- Select Hide Assigned. The unassigned controls appear in orange text.

To create connections between ATL controls and middleware-specific controls

- In the Wwise Controls pane, select and drag a control to the Connected Controls area of the Inspector pane.
To create a control

1. In the Wwise Controls pane, select and drag a middleware control to the ATL Controls pane.

   This creates a new control, which shares the same name of the middleware control. The middleware control and the ATL control are also automatically connected.

2. To preview the control, choose File, Save All.

ATL Controls

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The game uses Audio Translation Layer (ATL) controls to communicate with the audio middleware. ATL controls are mapped to various data authored in the middleware's authoring tool. This abstraction layer provides you the flexibility to change mappings quickly without updating the game's integration of controls.

To see the list of audio control types, see ATL Controls Pane (p. 3143).

ATL Default Controls

The Audio Controls Editor automatically creates the following ATL controls by default. You can find the controls in the default_controls folder.
# Default ATL Controls

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>do_nothing</strong></td>
<td>Trigger that is used as a blank event where play/stop trigger pairs can be assigned. If you set do_nothing on the stop trigger, the play trigger does not stop automatically.</td>
</tr>
<tr>
<td><strong>get_focus</strong></td>
<td>Trigger that is called when the application window in Lumberyard Editor gains focus.</td>
</tr>
<tr>
<td><strong>lose_focus</strong></td>
<td>Trigger that is called when the application window in Lumberyard Editor loses focus.</td>
</tr>
</tbody>
</table>

**Note**
To disable the `get_focus` and `lose_focus` triggers, use the console command `s_IgnoreWindowFocus = 1`. This is useful when remote connecting Wwise Profiler so that audio continues to play while the Wwise Authoring Tool application has focus.
### Name | Description
--- | ---
mute_all | Trigger that is called when you click **Mute Audio**, which is located on the lower menu bar of Lumberyard Editor.
unmute_all | Trigger that is called when you click **Mute Audio**, which is located on the lower menu bar of Lumberyard Editor.

**object_speed** | RTPC control that is updated according to the speed of the associated entity in the level. You can enable the calculation of speed on a per entity basis with the **object_velocity_tracking** control.

**object_velocity_tracking** | Switch that is used to enable or disable the calculation of the **object_speed** value on a per entity basis. You do not need to connect this switch to the audio middleware because it communicates Lumberyard-specific data.

**ObstructionOcclusionCalculationType** | Switch that is used to set the obstruction and occlusion calculation method of an entity. The switch state values are **Ignore**, **SingleRay**, and **MultiRay**. You do not need to connect this switch to the audio middleware because it communicates Lumberyard-specific data.

---

**Audiokinetic Wwise**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard supports two flavors of Audiokinetic's Wwise audio middleware technology:

- **Wwise** – The full product contains advanced features, supports third-party plugins, has full surround capabilities, and more. Wwise must be purchased separately.
  
  For information, see https://www.audiokinetic.com/pricing/ and https://www.audiokinetic.com/licensing/faq/.

- **Wwise LTX** – An exclusive, free version of Wwise is included with Lumberyard. Wwise LTX has a simplified authoring workflow and is a great starting point for new users.
  
  For a comparison of features, see Comparing Wwise Full to Wwise LTX (p. 3154).

Wwise software consists of the SDK and the Authoring Tool. The SDK contains the headers and libraries that are linked with the Lumberyard engine code. The Authoring Tool is the application software for editing, tweaking, testing, and profiling sounds for your project.

**Topics**

- Setting up Wwise LTX (p. 3150)
- Setting up a Wwise Project (p. 3151)
Setting up Wwise LTX

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard includes Audiokinetic Wwise LTX, which is an exclusive, free version of the Audiokinetic Wwise audio middleware. Sound designers and composers can use Wwise LTX to work independently from the engineering team and author rich soundscapes for your games.

If your game requires advanced features, you can upgrade to the full version of Wwise. For more information, see Upgrading Wwise LTX to the Full Version (p. 3158).

Topics
- Installing Wwise LTX Authoring Tool (p. 3150)
- Running the Wwise LTX Authoring Tool (p. 3151)
- Accessing Wwise LTX Documentation (p. 3151)

Installing Wwise LTX Authoring Tool

To author sounds with Wwise LTX for your game, you must first install Wwise LTX Authoring Tool.

To install Audiokinetic Wwise LTX Authoring Tool

1. Run Lumberyard Setup Assistant (p. 16).
2. Choose Install software.
3. Locate Audiokinetic Wwise LTX Authoring Tool entry under Optional software and choose Install it. This runs the Wwise Launcher.
4. If prompted to sign in to your Audiokinetic account, provide the requested information and choose Log in, or choose Sign up now to create an account.
5. Select the desired installation components and settings for Wwise LTX or accept the default, and then choose Next.

   Note
   The Wwise LTX SDK is already included in Lumberyard's 3rdParty directory. If you prefer a lightweight install, you can clear the SDK component and all the deployment platforms.

6. Review and update the installation Target Directory as needed, then choose Install.
7. Review the license terms, then choose Accept if you wish to proceed with installation.
8. Once the installation successfully completes, close the Wwise Launcher and return to Lumberyard Setup Assistant. It should now show that Wwise LTX is installed. You can now close Lumberyard Setup Assistant.

   Note
   Lumberyard does not require installation of the new version of Wwise listed in the Wwise Launcher.
Running the Wwise LTX Authoring Tool

To run the Wwise LTX Authoring Tool, you must first open or create a project. The Samples Project and Starter Game both include a Wwise LTX audio project that you can use.

**To run the Wwise LTX Authoring Tool from Wwise Launcher**

1. Run **Wwise Launcher**.
2. Choose the **Wwise** tab, locate the LTX installation and choose **Launch Wwise (64-bit)**. If preferred, choose the wrench icon to open a menu and choose **Create Desktop Shortcut** for quicker launch of the authoring tool next time.
3. If this is your first time running Wwise LTX, you are prompted again to review and accept the End-User License Agreement (EULA). After accepting the EULA, choose **Open Other** in the **Project Launcher** dialog box. With successive use of the Wwise Launcher, you will be able to select from recently opened Wwise projects and choose **Open Selection**.
4. Browse to one of the following directories, select the `.wproj` file that you find there, and choose **Open**:
   - `lumberyard_version\dev\SamplesProject\Sounds\wwise_project`
   - `lumberyard_version\dev\StarterGame\Sounds\wwise_project`
5. Alternatively, if you are not using the Samples Project or the Starter Game project, you can choose **New** in the **Project Launcher** dialog box to create a Wwise LTX project.

For more information about setting up Wwise for your game project, see **Setting up a Wwise Project** (p. 3151).

Accessing Wwise LTX Documentation

You can access the Wwise LTX documentation in the Wwise Authoring Tool.

**To access Wwise LTX documentation**

1. Run the **Wwise LTX Authoring Tool**.
2. On the **Help** menu, choose **Wwise Help**.

Setting up a Wwise Project

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The **Audio Controls Editor** looks for Wwise project files in the `lumberyard_version\dev \game_project\Sounds\wwise_project\` directory.

Lumberyard looks for the Wwise Soundbanks in the `lumberyard_version\dev \game_project\Sounds\wwise\` directory.

Creating a Wwise Project

**To create a Wwise project**

1. From the Start menu, open **Wwise Launcher**.
2. On the WWISE tab, click **Launch Wwise (64-bit)**.

   **Note**  
   If you have multiple versions of Wwise installed, open the version that you installed for Lumberyard.

3. In the **Project Launcher** dialog, click **New**.

4. In the **New Project** dialog, enter a **Name** for the project.

   **Note**  
   Wwise uses the name of your project to create a directory with the same name. However, you later rename the directory `wwise_project`.

5. For the **Location**, select the `lumberyard_version\dev\game_project\Sounds` directory. If this directory doesn't exist, create it.

6. Review and adjust the remainder of the options and then click **OK**.

7. If you are using the full version of Wwise, the **License Manager** dialog appears. Paste or import your license key if you have one and click **Save**.

   If you don't have your license key, click **Close**.

8. Exit Wwise.

9. In a file explorer, navigate to the `lumberyard_version\dev\game_project\Sounds` directory.

10. Rename the project directory you created in step 4 to `wwise_project`.

11. Create another directory next to `wwise_project` called `wwise`, if it doesn't exist already. This is where your Wwise soundbanks will be generated.

12. Open Wwise again.

13. Click **Open Other**.

14. Navigate to the `lumberyard_version\dev\game_project\Sounds\wwise_project` directory.

15. Select the `.wproj` file and click **Open**.

---

**Generating New Soundbanks**

**To generate new soundbanks**

1. If the Wwise project is not currently open, open it.

2. Choose **Project, Project Settings**.

3. On **SoundBanks** tab, in the **SoundBank Paths** section, edit the **SoundBank Folder** and select the `wwise` directory that you created in Creating a Wwise Project (p. 3151) and then click **OK**.

4. If you are using Wwise LTX, click **Generate Soundbank**.

   If you are using the Wwise full version:

   a. Press F7 to open the **SoundBank Manager Layout**.

   b. If no soundbanks exist, you must create one.

   To create a soundbank, select all **SoundBanks, Platforms**, and **Languages**.

   c. Click **Generate**.

5. Ensure that the soundbank generation succeeded. To do this, navigate to the `lumberyard_version\dev\game_project\Sounds\wwise` directory and verify that you have an `Init.bnk` file and other `.bnk` or `.wem` files.

For information on how to map soundbanks to ATL preloads so that Lumberyard can load them, see Using the Audio Controls Editor (p. 3142).
Creating Wwise Events for the Default ATL Controls

To create Wwise events for the default ATL controls

1. If the Wwise project is not currently open, open it.
2. Select the Events tab.
3. Under the Default Work Unit, select or create a directory to contain default ATL controls.
4. Right-click the directory to display the context menu and choose New Child, Empty Event.
5. Create five empty events named the following:
   - do_nothing
   - get_focus
   - lose_focus
   - mute_all
   - unmute_all

   **Note**
   - The get_focus and unmute_all events unmute the Master Audio Bus.
   - The lose_focus and mute_all events mute the Master Audio Bus.
6. Save the project.
8. In Lumberyard Editor, open the Audio Controls Editor.
9. Map these Wwise events to the default ATL controls.
10. Click File, Save All.

Connecting Wwise to the Editor and Game

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can establish a remote connection between Wwise and Lumberyard for your project's debug and profile builds of Lumberyard. By default, this feature is disabled. Remote connecting is not supported for performance and release builds.

**To enable remote connection**

1. In a text editor, create a file in the lumberyard_version\dev directory and name it user.cfg.
2. Open the file and enter the following parameter and value: s_WwiseEnableCommSystem=1.
3. Save the file.
4. Open your game or Lumberyard Editor.
   **Note**
   If your game or Lumberyard Editor is already running, close and reopen it.
5. Open the Wwise Authoring Tool and open the Wwise project associated with your game.
6. In the Wwise Authoring Tool, click the Connect to Remote Platform icon in the toolbar: 

   The Remote Connections dialog displays a list of computers to which you can connect.
7. Select the **Localhost** entry and click **Connect**. To verify a successful connection, look for the "Connected to" text in the Wwise toolbar.

**Note**

Localhost is the running instance of the game or Lumberyard Editor on your computer with remote connection enabled.

8. You can live edit parameters, mute or solo objects or sounds, and tweak effects in the Wwise project. Because remote connection is engaged, you can hear the changes instantly. Remote connecting is useful not only for profiling but also for debugging.

For additional information and useful tips on using the remote connection, refer to the documentation included with Wwise.

### Comparing Wwise Full to Wwise LTX

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The following comparison table summarizes the differences in features between Wwise LTX and Wwise Full.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Wwise</th>
<th>Wwise LTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound SFX</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sound Voice</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sound Voice</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Actor Mixer</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Virtual Folders</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Work Units</td>
<td>Yes</td>
<td>Default Only</td>
</tr>
<tr>
<td>Physical Folders</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Audio Bus</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No channel configuration</td>
</tr>
<tr>
<td>Motion Bus</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Auxiliary Bus</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interactive Music</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Events</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Some actions unavailable</td>
</tr>
<tr>
<td>Dialog Event</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sound Bank Management</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Switches</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>States</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Feature</td>
<td>Wwise</td>
<td>Wwise LTX</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No property binding</td>
</tr>
<tr>
<td>Game Parameters</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Effects Share Sets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Limited list of effects</td>
</tr>
<tr>
<td>Attenuations Share Sets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Conversion Settings Share Sets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modulators Share Sets</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Soundcaster Sessions</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mixing Desk Sessions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Control Surface Sessions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Queries</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Override Parent</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Surround Sound</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stereo only</td>
</tr>
<tr>
<td>RoomVerb Effect</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Delay Effect</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Parametric EQ Effect</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All Other Effects</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Audio Input Source</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sine Source</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Tone Generator Source</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Silence Source</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Motion Generator Source</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Synth One Source</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>External Sources</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>REV Plugin</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Available with license</td>
</tr>
<tr>
<td>SoundSeed Air / Impact Plugins</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Available with license</td>
</tr>
<tr>
<td>Convolution Reverb Plugin</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Available with license</td>
</tr>
<tr>
<td>Feature</td>
<td>Wwise</td>
<td>Wwise LTX</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>User-Defined Auxiliary Send Volumes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Game-Defined Auxiliary Send Volumes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bus Volume</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Voice Volume</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Voice Pitch</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Voice Low-Pass Filter</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Voice High-Pass Filter</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All Sends Properties</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Output Bus Volume/Low-Pass/High-Pass</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Bus Effect</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Conversion Settings</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Initial Delay</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2D / 3D</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Attenuation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Panner 2D Slider</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RTPC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Except for LFO, Envelope, or MIDI parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perforce / SVN</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Property Editor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>A subset of the tabs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Editor - General Settings</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Editor - Source Settings</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Editor - Effects</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Property Editor - Positioning</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### Comparing Wwise Full to Wwise LTX

<table>
<thead>
<tr>
<th>Feature</th>
<th>Wwise</th>
<th>Wwise LTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Editor - RTPC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Property Editor - States</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Property Editor - HDR</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Property Editor - Mixer-Plugin</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Property Editor - Motion</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Property Editor - MIDI</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Property Editor - Advanced Settings</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Transport</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Soundcaster</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Contents Editor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>List View</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Query Editor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Reference View</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multi-Editor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Project Explorer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Conversion Settings</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Effect Editor / Source Editor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modulation Editor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Event Editor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Event Viewer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote Connection</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Capture Log</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Advanced Profiler</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Game Object 3D View</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Game Object Explorer</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Performance Monitor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Voice Monitor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Profiler Statistics</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### Upgrading Wwise LTX to the Full Version

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To upgrade Wwise LTX to the full version of Wwise, follow the procedures in this topic.

**Topics**
- Upgrading the SDK (p. 3158)
- Upgrading Your Project (p. 3159)
- Upgrading Your Soundbanks (p. 3160)

After you upgrade the SDK, project, and soundbanks, you can open your game in Lumberyard to test your sounds.

**Upgrading the SDK**

**To upgrade the SDK**

1. Open **Wwise Launcher**. If you don't have it, install it from https://www.audiokinetic.com/download.
2. On the **WWISE** tab, under **Install New Version**, click the dropdown menu, and select **Wwise 2018.1.4.6807**, which is the version supported by Lumberyard.
3. Click **Install**.
4. For **Packages**, select **SDK (C++)**.
5. For **Deployment Platforms**, select the platforms required for your project.

Some platform components are available only after you purchase them from Audiokinetic. For more information about licensing, see https://www.audiokinetic.com/pricing.

---

<table>
<thead>
<tr>
<th>Feature</th>
<th>Wwise</th>
<th>Wwise LTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schematic View</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Attenuation Editor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Position Editor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Control Surfaces</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Integrity Report</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Loudness Meter</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Meter</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MIDI Keymap Editor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mixing Desk</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Music Views</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SoundBank Views</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
6. Enter the installation directory and click **Next**.
7. Select the plugins to install, and click **Install**.
8. After installation completes, navigate to the \lumberyard_version\3rdParty\Wwise directory.
9. Create a directory named **2018.1.4.6807**.
10. Navigate to the Wwise installation directory that you specified in step 6, and copy the SDK directory into the \lumberyard_version\3rdParty\Wwise\2018.1.4.6807 directory.
11. To verify that Lumberyard recognizes the Wwise installation and configuration:
   a. Run Lumberyard Setup Assistant.
   b. On the **Install optional SDKs** page, verify that a green checkmark appears next to **Audiokinetic Wwise**.
12. Navigate to the \lumberyard_version\dev\Gems\AudioEngineWwise\Code directory.
13. In a text editor, open the *wscript* file.
14. Edit the *wwise_flavor* parameter from **WWISELTX** to **WWISE** and save the file.
   This instructs the WAF build system to use Wwise instead of Wwise LTX.
15. Configure WAF and build the project. For more information, see the section called “Build a project” (p. 61).

**Upgrading Your Project**

After you successfully configure and build Lumberyard, you can upgrade the content in your project. This assumes that you already have a Wwise LTX project set up for your game project.

Follow this procedure to upgrade an existing Wwise LTX project to use the full version of Wwise.

**Prerequisites**

If your Wwise LTX project and soundbanks are under source control, you must check them out. Check out the entire \lumberyard_version\dev\game_project\Sounds directory. After you upgrade your project, select the files that were created, and then revert any unmodified files before you submit them.

**Important**

During the upgrade process, read each message that Wwise displays.

**To upgrade your project**

1. Open **Wwise Launcher**.
2. On the **WWISE** tab, in the **Versions Installed** list, find **2018.1.4.6807**. Verify that the full revision is listed.
3. Click **Launch Wwise (64-bit)**.
4. If this is the first time running Wwise Authoring Tool for this version, an End-User License Agreement (EULA) dialog appears. Review and accept the EULA.
5. In the **Project Launcher** dialog, click **Open Other**, and open the *file_name*.wproj file for your game project.

   You can find this file in the \lumberyard_version\dev\game_project\Sounds \wwise_project directory.
   • If a message appears stating that you are upgrading from an earlier version of Wwise, you must upgrade the Wwise project XML schema. The message lists the files to be updated and highlights any files that are read-only.
• The message also states that this action is a one-way project upgrade. This means that the
upgraded project no longer loads with Wwise LTX. You can back up of your Wwise LTX project
before you click Yes in this message.
• The Project Load Log message displays all the changes made. You might see information about
default work units that were missing and created when you upgraded.
• If your Wwise project is under source control, ensure that you add these .wwu files to your depot.
6. Close the message and save the Wwise project.

Wwise LTX uses a single generic platform. After upgrading to Wwise, you can edit the platforms for your
project.

Upgrading Your Soundbanks

After you upgrade the SDK and your project, you can rebuild your soundbanks.

Note
Wwise LTX uses a single Generic platform. You can edit the platforms by going to Project,
Platform Manager. If you change the platforms, you must reopen your project.

To upgrade your soundbanks
1. Open your project in Wwise.
2. Click Project, License Manager.
3. Choose to either import your license or paste it from the clipboard.
4. Click Save.
5. Navigate to Project. Click Project Settings or press Shift+K, and select the SoundBanks tab.
6. For each platform in your project, click the ellipsis to open the Post-Generation Step editor.
   a. In the Post-Generation Step editor, click Load, From Last Location.
   b. Navigate to the lumberyard_version\dev\Tools\WwiseAuthoringScripts directory,
      select the ly_copy_output_and_generate_metadata.wcmdline file, and click Open.
   c. Click OK in the Post-Generation Step editor.
7. Click OK in the Project Settings window.
8. Open the SoundBank Layout. Press F7 or click Layouts, Soundbank.
10. Click Generate. The Generating SoundBanks dialog displays the progress.

The most common errors encountered during the generate process are caused by read-only files or
by not having a license applied to the project.

Lumberyard is now using the full version of Wwise. You can now run Lumberyard Editor and your game
to test your sounds.

Using a Different Wwise Version

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

You can also use an older or newer version of Wwise. However, Lumberyard no longer supports versions 2016 and earlier.

**Configuring an Earlier Version of Wwise for Lumberyard**

Follow this procedure to use the 2017 version of Wwise.

**To configure Lumberyard for the 2017 version of Wwise**

1. Navigate to the `{lumberyard_version}/dev/Gems\AudioEngine\Wwise\Code` directory.
2. In a text editor, open the `wwise_backwards_compatibility.README.txt` file.
3. Follow the instructions in the file. The following is an overview of the instructions.
   a. Download a 2017 version of Wwise.
   b. Copy the `{Wwise_version}/SDK` directory from the Wwise install directory to the `{lumberyard_version}/3rdParty/Wwise` directory.
   c. In a text editor, open the `{lumberyard_version}/dev\SetupAssistantConfig.json` file and modify the version of Wwise in the file.

   This configures Lumberyard Setup Assistant to look for that specific version instead.
   d. Add a `.json` file that defines the Wwise 2017 SDK for WAF.
4. After you complete these steps, run Lumberyard Setup Assistant and the `lmbr_waf configure` command to check for errors.

   If no errors occur, build Lumberyard Editor and Engine for Wwise 2017.

**Configuring a Later Version of Wwise for Lumberyard**

Audiokinetic frequently releases new versions of Wwise. Later versions might be compatible with the current version supported by Lumberyard without requiring major code changes. However, minor revisions can contain API updates that require code changes in Lumberyard. Major revisions can require code changes and deeper migration.

If you want to switch to later versions of Wwise, use the following procedure.

**To configure Lumberyard for a later version of Wwise**

1. Download the version of Wwise that you want.
2. Copy the `{Wwise_version}/SDK` directory from the Wwise install directory to the `{lumberyard_version}/3rdParty/Wwise/your_new_wwise_version` directory.
3. In a text editor, open the `{lumberyard_version}/dev\SetupAssistantConfig.json` file and replace all occurrences of `2018.1.4.6807` with the version that you downloaded.

   This configures Lumberyard Setup Assistant to look for the version that you specify.

   `SetupAssistantConfig.json` contains a section for `Wwise` and a separate section for `Wwise LTX`. Modify only the `Wwise` section.
4. Save and close the `SetupAssistantConfig.json` file.
5. Run Lumberyard Setup Assistant. Then verify that the new Wwise SDK version is configured correctly on the Install optional SDKs page.

   You’ll notice that the setup instructions Lumberyard Setup Assistant still mention Wwise 2018.1.4.6807. You can ignore this text.
6. Complete the steps to upgrade Wwise (p. 3158).
Audio Components

Use the following audio components to add and configure sounds in your level:

- **Audio Area Environment** (p. 551) – Enables entities that are moving around and throughout a shape to have environment effects applied to any sounds that they trigger. You must also add a shape component to use the audio area environment component.
- **Audio Environment** (p. 552) – Provides access to features of the Audio Translation Layer (ATL) environments. Environments are used to apply environmental effects such as reverb or echo.
- **Audio Listener** (p. 553) – Places a virtual microphone in the environment. An audio listener acts as a sink for sound sources in the virtual world, and 3D audio rendering is processed with respect to the listener's world transform. You can specify the audio listener's position and rotation independently.
- **Audio Preload** (p. 555) – Loads and unloads ATL preloads, which contain references to soundbanks.
- **Audio Proxy** (p. 557) Required dependency if you add multiple audio components to an entity. It acts as a proxy audio object wrapped in a component. For example, if you have an audio trigger component and an audio rtpc component on the same entity, they communicate to the same audio object using this audio proxy component.
- **Audio RTPC** (p. 557) Provides basic Real-Time Parameter Control (RTPC) functionality. An RTPC is a named variable that the audio system can interpret in many different ways. It allows game developers to set the value from the game at run time to produce real-time tweaking of sounds.
- **Audio Switch** (p. 558) – Provides basic Audio Translation Layer (ATL) switch functionality. With switches (and switch states), you can specify the state of an entity. The audio middleware interprets states, modifies the behavior of sounds, and plays the appropriate sounds.
- **Audio Trigger** (p. 559) – Provides basic play and stop features so that you can set up Audio Translation Layer (ATL) play and stop triggers that can be executed on demand. With an audio trigger, you can also enable the player to run or stop audio triggers by name on entities.

Related topics:

- Microphone Gem (p. 1171)
- Cloud Canvas Cloud Gems (p. 1122)
- Text to Speech Cloud Gem (Using Amazon Polly) (p. 2203)
- Speech Recognition Cloud Gem (p. 2203)

Audio Console Variables

The following console variables can be used with the Lumberyard Audio system.

**s_ATLPoolSize**

Specifies in KB the size of the memory pool to be used by the audio translation layer (ATL).
**Audio Console Variables**

Default values: PC = 8192, Mac = 8192, Linux = 8192, iOS = 8192, Android = 4096

**s_AudioEventPoolSize**

Sets the number of preallocated audio events.

Default values: PC = 512, Mac = 512, iOS = 128, Android = 128

**s_AudioLoggingOptions**

Toggles the logging of audio-related messages.

Default values: 0 (disabled), a = Errors, b = Warnings, c = Comments

**s_AudioObjectsDebugFilter**

Allows for filtered display of audio objects by a search string.

Default value: "" (all)

**s_AudioObjectPoolSize**

Sets the number of preallocated audio objects and corresponding audio proxies.

Default values: PC = 2048, Mac = 2048, iOS = 256, Android = 256

**s_AudioProxiesInitType**

Can override on a global scale. If set, it determines whether AudioProxies initialize synchronously or asynchronously. This is a performance variable, as asynchronously initializing AudioProxies has a greatly reduced impact on the calling thread. When set to initialize asynchronously, audio playback is delayed.

Values: 0 = AudioProxy-specific initialization; 1 = Initialize synchronously; 2 = Initialize asynchronously.

Default value: 0 (all platforms)

**s_AudioTriggersDebugFilter**

Allows for filtered display of audio triggers by a search string.

Default value: "" (all)

**s_DrawAudioDebug**

Draws AudioTranslationLayer related debug data to the screen.

Values:
- 0: No audio debug info on the screen
- a: Draw spheres around active audio objects
- b: Show text labels for active audio objects
- c: Show trigger names for active audio objects
- d: Show current states for active audio objects
- e: Show RTPC values for active audio objects
- f: Show Environment amounts for active audio objects
- g: Draw occlusion rays
- h: Show occlusion ray labels
- i: Draw sphere around active audio listener
- v: List active Events
- w: List active Audio Objects
- x: Show FileCache Manager debug info
Audio Console Variables

- `y`: Show memory pool usage info for the audio impl

**s_ExecuteTrigger**

Executes an Audio Trigger. The first argument is the name of the audio trigger to be executed, the second argument is an optional audio object ID. If the second argument is provided, the audio trigger is executed on the audio object with the given ID; otherwise, the audio trigger is executed on the global audio object.

**s_FileCacheManagerDebugFilter**

Allows for filtered display of the different AFCM entries such as Globals, Level Specifics, and Volatiles.

Values: Default = 0 (all); a = Globals; b = Level Specifics; c = Volatiles

**s_FileCacheManagerSize**

Sets the size in KB that the AFCM allocates on the heap.

Default values: PC = 393216, Mac = 393216, Linux = 393216, iOS = 2048, Android = 73728

**s_FullObstructionMaxDistance**

For sounds whose distance to the listener is greater than this value, the obstruction value is attenuated with distance.

Default value: 5 m

**s_IgnoreWindowFocus**

If set to 1, the sound system continues to play when the Editor or Game window loses focus.

Default value: 0 (off)

**s_OcclusionMaxDistance**

Obstruction/Occlusion is not calculated for the sounds whose distance to the listener is greater than this value. Set this value to 0 to disable obstruction/occlusion calculations.

Default value: 500 m

**s_OcclusionMaxSyncDistance**

Physics rays are processed synchronously for the sounds that are closer to the listener than this value, and asynchronously for the rest (possible performance optimization).

Default value: 10 m

**s_PositionUpdateThreshold**

An audio object has to move by at least this amount to issue a position update request to the audio system.

Default: 0.1 (10 cm)

**s_SetRtpc**

Sets an Audio Rtpc value. The first argument is the name of the audio RTPC, the second argument is the float value to be set, the third argument is an optional audio object ID. If the third argument is provided, the RTPC is set on the audio object with the given ID. Otherwise, the RTPC is set on the global audio object.

**s_SetSwitchState**

Sets an audio switch to a provided state. The first argument is the name of the audio switch, the second argument is the name of the switch state to be set, the third argument is an optional audio
object ID. If the third argument is provided, the audio switch is set on the audio object with the given ID; otherwise, the audio switch is set on the global audio object.

**s_ShowActiveAudioObjectsOnly**

When drawing audio object names on the screen, this variable is used to choose between all registered audio objects or only those that reference active audio triggers.

Default value: 1 (active only)

**s_StopTrigger**

Stops an audio trigger. The first argument is the name of the audio trigger to be stopped, the second argument is an optional audio object ID. If the second argument is provided, the audio trigger is stopped on the audio object with the given ID; otherwise, the audio trigger is stopped on the global audio object.

**s_VelocityTrackingThreshold**

An audio object has to change its velocity by at least this amount to issue an `object_speed` RTPC update request to the audio system.

Default value: 0.1 (10 cm/s)
Platform development in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Learn about the platforms supported by Lumberyard, their specific requirements, and how to test, build, and ship your project. In addition to Windows support, Lumberyard runs on the iOS and Android mobile platforms, macOS, and as dedicated Linux servers. This section also covers building for and running on virtual reality headsets. This documentation covers design considerations for all of these platforms, how to build and deploy, and debugging and troubleshooting advice. Each platform also has documentation specific to its own requirements.

Topics
- Developing for Android and iOS with Lumberyard (p. 3166)
- Create virtual reality projects in Lumberyard (p. 3248)
- Create macOS projects in Lumberyard (p. 3263)
- Creating Lumberyard Executables for Linux (p. 3269)

Developing for Android and iOS with Lumberyard

Amazon Lumberyard comes with support for building and deploying your game to Android and iOS devices. In this section of the guide, you'll learn how to get your projects up and running on mobile, what are some of the important design considerations for these platforms, and which unique workflows are required. You'll also learn how to use the Virtual File System (VFS) server to serve assets on-demand to your game that's running on a physical device, and how to get started using Device Farm for large-scale testing of your mobile project.

Take a deep dive into the Android and iOS documentation to learn about the requirements and configuration to get started with these platforms. When you're up and running, look through the rest of the topics in this section to finish generic setup and take the next steps to make your Lumberyard project a great experience on mobile.

Topics
- Lumberyard Android support (p. 3167)
- iOS Support (p. 3195)
- Design Considerations for Creating Mobile Games Using Lumberyard (p. 3214)
- Lumberyard Performance Tuning Guidelines for Mobile Devices (p. 3215)
- Modifying Project Settings for Mobile Device Games (p. 3224)
- Updating Graphics Settings for Android and iOS (p. 3234)
Lumberyard Android support

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

In this section, you'll learn about the system requirements for Android. You will also learn how to set up your Amazon Lumberyard installation for Android builds, configure your project for Android support, and build and deploy to devices. When you finish your development cycle and are ready to release, that's covered here too. Along the way, you'll want to use the troubleshooting guide and reference materials to help resolve any issues you might find.

Topics
- Minimum device requirements (p. 3167)
- Prerequisites (p. 3167)
- Video tutorial (p. 3168)
- Set up your environment to develop for Android with Lumberyard (p. 3168)
- Configure Lumberyard projects for Android (p. 3172)
- Build and deploy your project for Android (p. 3175)
- Releasing Lumberyard projects for Android (p. 3180)
- Troubleshooting Lumberyard issues on Android (p. 3185)
- Reference for Android (p. 3187)

Minimum device requirements

Lumberyard has the following minimum requirements to deploy to Android. These are only minimum device specs. Newer devices offer better performance and require fewer optimizations. Most devices released in 2017 or later meet these criteria.

<table>
<thead>
<tr>
<th>CPU</th>
<th>ARMv8 quad-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>OpenGL ES 3.0 or 3.1 support</td>
</tr>
<tr>
<td>OS</td>
<td>Android 5.0 (API level 21)</td>
</tr>
</tbody>
</table>

If your Lumberyard project requires higher minimum specifications than these to achieve what you consider acceptable performance, you can set your own device and capability requirements through the Google Play Store. See the Android documentation for more information.

Prerequisites

To get started with your Lumberyard project on Android, you need the following software installed, in addition to Lumberyard. If you need to, download Lumberyard now (p. 10). Lumberyard can be configured for Android development either when you install it, or after you set up your Android tools. See Set up your environment to develop for Android with Lumberyard (p. 3168) for more information.
• An Android development environment. This can be either Android Studio or the standalone command line tools. For developers not already familiar with working with Android, we recommend Android Studio, which includes the command-line tools. Download Android developer tools.

• The required minimum development SDKs:
  • Android NDK r21
  • Android SDK API level 28 (Android 9.0 – Pie). The following libraries and tools are required as part of the SDK installation.
    • Google Play APK Expansion Library
    • Google Play Licensing Library
    • Android SDK Tools
    • Android SDK Platform-Tools
    • Build Tools 26.x.x or later

Important
Google requires that new apps submitted to the Google Play Store are built against an SDK version that is no more than one year old. To make sure that you meet this requirement, build against the latest available Android SDK and check for updated versions before any submission to the Google Play Store.

For instructions on installing the Android SDK and NDK, see the Android Studio SDK Manager or Command-line sdkmanager documentation on the Android developer site. For information on the packages required as part of the Android SDK install, see Set up your environment to develop for Android with Lumberyard (p. 3168).

Video tutorial

To help you get up and running quickly with Android, we have a video tutorial (10:10) that walks through the process of getting Android Studio installed, configuring your Lumberyard project, and running the project on an Android device.

Watch the video on YouTube: Setup Lumberyard to Build Android Games and Projects

Set up your environment to develop for Android with Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This section walks you through the steps needed to get your development environment and Amazon Lumberyard projects ready for build and deployment to Android. To get started, make sure that you have the Prerequisites (p. 3167). To learn about the platform-specific steps involved in project configuration and building, see Configure Lumberyard projects for Android (p. 3172) and Build and deploy your project for Android (p. 3175).

Topics
• Lumberyard environment setup (p. 3169)
• Add Android tools to PATH (p. 3171)
• Build system setup (p. 3171)
• Next steps (p. 3172)
Lumberyard environment setup

After you've installed the prerequisites (p. 3167), configure your Lumberyard installation to allow Android builds.

To allow Android builds

1. Start the Lumberyard Setup Assistant. If you're installing Lumberyard for the first time, Setup Assistant starts automatically. Otherwise, launch it from `lumberyard_install_dir\dev\Tools\LmbrSetup\Win\SetupAssistant.exe`.

2. Using Setup Assistant, select the following capabilities for your Lumberyard configuration:
   - Create, Modify and Build projects
   - Compile for Android Devices

   After selecting these capabilities, select Next to continue with configuration in the Setup Assistant.

3. Provide the paths to the Android Studio installation, and the `ndk-build.cmd` and `adb.exe` Android tools commands. These paths are used to determine the location of all of the Android build tools that Lumberyard uses. If you performed a default install of Android Studio and its tools, Setup Assistant should have already detected the installation paths.

   If you used another install location, or Setup Assistant couldn't find your Android install, provide the paths to the appropriate tools. You can also select the Browse button to use Windows Explorer to navigate to the required files.

   By default, the tools are located at the following paths.
   - Android Studio installation — `C:\Program Files\Android\Android Studio`
   - `ndk-build.cmd` — `C:\Users\username\AppData\Local\Android\Sdk\ndk\version\ndk-build.cmd`
   - `adb.exe` — `C:\Users\username\AppData\Local\Android\Sdk\platform-tools\adb.exe`

   **Tip**

   If you can't find your Android SDK install in `C:\Users\username\AppData\Local\Android\Sdk`, the SDK location might have been changed in Android Studio. To find out the location for your Android SDKs, follow these steps in Android Studio 4.0.x:
   - Open the Settings pane in Android Studio, either by navigating to the File > Settings menu from an open project, or by selecting Settings from the startup window.
   - In the left navigation pane of the Settings window, select Appearance & Behavior > System Settings > Android SDK.
• Note the path displayed as the **Android SDK Location**. Use this as the base path for locating the Android tools.

![Android SDK Location](image)

When each Android tool is marked with a green checkmark icon, you can close the Setup Assistant.

4. Check that the API level for the Android SDK you're using is correctly set in the Lumberyard build settings.
   a. Open the file `lumberyard_install_dir\dev\_WAF\_android\android_settings.json` in a text editor.
   b. Change the value of `SDK_VERSION` to `android-apilevel`, and `NDK_VERSION` to `android-ndk-version`. Use the API level of your development SDK for `apilevel`, and the API version of the NDK for `ndk-version`.

   For example, for Android Q (API level 29) and NDK API version 21, your configuration file will be similar to the following.

   ```json
   {
     "DEV_KEYSTORE_ALIAS" : "development_keystore",
     "DEV_KEYSTORE" : "_WAF_/android/dev.keystore",
     "DISTRO_KEYSTORE_ALIAS" : "distribution_keystore",
     "DISTRO_KEYSTORE" : "_WAF_/android/distro.keystore",
     "BUILD_TOOLS_VER" : "latest",
     "SDK_VERSION" : "android-29",
     "NDK_PLATFORM" : "android-21",
     "BUILD_ENVIRONMENT" : "Development"
   }
   ```

5. Configure the Lumberyard build system.
   a. Open a Windows command prompt and navigate to `lumberyard_install_dir\dev`.
   b. Run `lmbr_waf configure`.

At this point, if you're missing any of the required packages or tools, or if Lumberyard needs additional configuration, `lmbr_waf configure` will report an error telling you what you need to install or edit to get your build working. Use either **Android Studio SDK Manager** or **sdkmanager** to install any missing packages or SDKs.

**Important**
At this point, **don't** rebuild your game or engine. Wait until after you configure your project for Android, so that the Lumberyard tools will pick up those changes.
Add Android tools to PATH

When working with Android, it's useful to have access to the SDK tools directly from the command line. If you're working with build infrastructure or on a development team for your project, we recommend that you add the Android build tools to your PATH environment variable. To set an environment variable on Windows, follow these steps.

1. Open the Windows Control Panel and select System > Advanced system settings.
2. In the System Properties dialog box, select Environment Variables.
3. Under User variables, edit the PATH variable to add the directories containing the Android tools. Select the New button to add a new value to PATH. The values that you add are the directories containing the adb.exe and ndk-build.cmd files.

Build system setup

Now that you have Android support enabled in Lumberyard, you need to configure the build system to recognize your project and produce Android (es3) assets.

1. Open the file lumberyard_install_dir\dev\WAF\user_settings.options in a text editor.
2. In the [Game Projects] section of the configuration file, change the value of enabled_game_projects to include the project that you want to build for Android. This value setting is a comma-separated list of the projects to generate Android targets for during configuration. Either remove the existing values, or add your project name to the list. For example, to enable SamplesProject, MultiplayerSample, and your project:

   [Game Projects]
   enabled_game_projects = SamplesProject, MultiplayerSample, your-project-name

3. Save and close the file.
4. Open the file lumberyard_install_dir\dev\AssetProcessorPlatformConfig.ini in a text editor.
5. Remove the semicolon to uncomment es3=enabled.

   [Platforms]
   ;pc=enabled
   es3=enabled
   ;ios=enabled
   ;osx_gl=enabled

6. Save and close the file.
7. Start the Asset Processor to compile any existing assets for use on Android.
   a. Check to see if the Asset Processor is running. Open the Windows system tray and look for the Asset Processor icon.
b. If the Asset Processor is running, double-click the icon in the tray to bring up the Asset Processor window.

c. If the Asset Processor isn't running, launch it from `lumberyard_install_dir\dev\Bin64\vc142\AssetProcessor.exe`.

8. Wait until the Asset Processor finishes building all of your assets for use with Android. When assets are finished processing, the Asset Processor status line indicates that the Asset Processor is **Idle**.

**Next steps**

Now that you have your Lumberyard environment configured for Android, configure, build, and deploy your project to a device to test.

- Configure Lumberyard projects for Android (p. 3172)
- Build and deploy your project for Android (p. 3175)

**Configure Lumberyard projects for Android**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the [AWS Game Tech blog](#) to learn more.

In addition to the system-wide configuration requirements (p. 3168) to build for Android, Amazon Lumberyard requires some project-specific settings as well. These settings not only include things like supplying values to the Android Manifest, they also control the application's ability to connect to the shader compiler and how assets are loaded onto the device. With the Virtual File System (VFS) offered by the Asset Processor, you can even live-reload assets on the device as they change.

This section covers the basics of configuring your project so that you can work with Android. For all of the available settings, see Reference for Android (p. 3187).

**Important**
Because project settings impact how your application communicates with some services on the Asset Processor host and modify the Android manifest, changing any settings for your project will require a rebuild.

**Topics**
- Handling assets on Android (p. 3172)
- Serving assets over the Virtual File System (VFS) (p. 3174)
- Custom Android manifests (p. 3174)

**Handling assets on Android**

Working with Android in Lumberyard can be complicated because of the restrictions of the Android platform, which affects how Android application packages (APKs) are built and deployed. Different asset deployment configurations can greatly affect performance during your development process, so make sure that you select the one that's right for your use case. This section helps break down what options are available, when they're most appropriate, and how you configure your Lumberyard project in order to support each.

These are the asset modes that are available modes for asset deployment to devices:

- `configuration_default` — assets are packed according to the build configuration:
  - The **debug** and **profile** configurations use `loose_files` mode.
• The release and performance configurations use the project_settings mode to determine how to build assets, the APK, and any OBB files.
• loose_files and loose_paks — assets are loose, which means that they are copied to storage media. For more information about loose files, see Loose modes (p. 3173).
• apk_files and apk_paks — assets are packed, which means that they are bundled as part of your APK. For more information about packed files, see Packed modes (p. 3173).
• project_settings — assets are packed according to the configuration.

To configure which asset mode your deployments use, open lumberyard_install_dir\dev\WAF_\user_settings.options in a text editor and modify the [Android Options] section.

[Android Options]
android_asset_mode = [ configuration_default | loose_files | loose_paks | apk_files | apk_paks | project_settings ]

Loose modes

In loose files mode (loose_files or loose_paks), none of your assets are included into the APK. Instead, assets are copied directly to the storage media on the Android device and loaded by the Lumberyard engine at runtime. When using loose_paks, assets are first placed into a .pak file and then pushed to the device.

A loose files mode is probably what you want to be using for day-to-day development and to rapidly iterate on assets and deploy updates to the device.

When using loose assets, we recommend the loose_files mode to make manual inspection of the device's SD card easier and on-demand asset deployments faster, at the cost of a small performance hit when making the initial asset load. To configure your deployments to use loose files mode, open lumberyard_install_dir\dev\WAF_\user_settings.options in a text editor and modify the [Android Options] section.

[Android Options]
android_asset_mode = loose_files

Packed modes

Unlike many Lumberyard scenarios in which packed assets refer to .pak files, the packed assets (apk_files and apk_paks) modes bundle your assets as part of the deployed APK, with the option of putting them into a .pak first. This APK is sometimes referred to as a packed APK, which refers to the fact that assets are packed into the APK and not distributed separately.

Bundling your assets in an APK allows for deployment of a single artifact, making these modes ideal for building APKs to distribute for testing or previews.

When using packed assets, we recommend the apk_paks mode, which offers some performance improvements. To configure your deployments to use packed assets mode, open lumberyard_install_dir\dev\WAF_\user_settings.options in a text editor and modify the [Android Options] section.

[Android Options]
android_asset_mode = apk_paks

Tip
If you want to do large-scale automated testing across multiple devices, you can build packed APKs and distribute them to AWS Device Farm.
Serving assets over the Virtual File System (VFS)

During the early stages of development with Lumberyard, and for users rapidly iterating on assets in a way that requires live reloading, you can use the Virtual File System (VFS) to stream assets to a device from your development machine. The VFS uses the Asset Processor as a proxy that your Android device connects to in order to receive updated assets. Whenever the Asset Processor rebuilds a product asset, that asset is copied to the device and reloaded. The VFS doesn't do initial asset deployments — your build configuration's asset mode is respected — only reloads.

To turn on VFS

1. Open the file `lumberyard_install_dir\dev\bootstrap.cfg` in a text editor.
2. Turn on the Asset Processor VFS support by changing the `remote_filesystem` value to 1.

   ```
   remote_filesystem=1
   ```

3. Configure the connection information for the VFS by changing the values of `remote_ip`, `remote_port`, and `white_list`. For an Asset Processor running locally, both the remote IP address and allow list should be `127.0.0.1`. The default port of the Asset Processor is `45643`.

   ```
   remote_ip=127.0.0.1
   remote_port=asset_processor_port
   white_list=127.0.0.1
   ```

4. Configure the Lumberyard engine to connect to the VFS and boot without some assets preloaded.

   ```
   connect_to_remote=1
   wait_for_connect=0
   ```

5. Configure the connection from the device to the shader compiler. You can either do this by starting the Shader Compiler yourself, or by passing traffic to the Shader Compiler through the Asset Processor.

   - **Connect directly to shader compiler** – See Connect to the Shader Compiler (p. 3176) for instructions.
   - **Connect to Asset Processor** – Edit the Android configuration to use the Asset Processor's shader compiler process.

     a. Open the file `lumberyard_install_dir\dev\system_android_es3.cfg` in a text editor.

     b. Change the value of `r_AssetProcessorShaderCompiler` to 1.

       ```
       r_AssetProcessorShaderCompiler=1
       ```

6. Set up port forwarding from the device to the Asset Processor.

   ```
   adb reverse tcp:45643 tcp:45643
   ```

7. Rebuild your Lumberyard project for Android and deploy it to the device.

Now when you launch your Lumberyard project, it will connect to the Asset Processor host to do live asset reloads over the VFS.

Custom Android manifests

As part of each Android application, Google requires a list of activities that will be performed by the application, and which device capabilities are required. For example, if your project uses live location data
on an Android device, you'll need to ensure that your project has the appropriate permissions to collect
data at the right intervals and with the appropriate location precision. Because Lumberyard projects are
made up of many components, during the build process, the build tools pick up multiple manifests and
merge them together.

If you're using a gem or other component that requires special device access, or that runs its own Android
Activity or Intent, the gem requires an Android manifest file. This file needs to meet the following criteria
to be picked up by the build system:

- The file must be named `AndroidManifest.xml` and conform to the Android manifest format.
- The gem or module's `wscript` build file must include `android_manifest_path` as a key in the gem
  definition that's passed to the build system. The value of this key should be the absolute path to the
directory containing the manifest.

For example, the microphone gem (p. 1171) uses an Android manifest to give your project
access to the microphone. Below is a portion of the microphone gem's `wscript` that defines
how to locate the Android manifest file. This snippet sets up the gem to use the manifest
at `lumberyard_install_dir\dev\Gems\Microphone\Code\Source\Android
\AndroidManifest.xml`.

```python
def build(bld):
    manifest_src_root = bld.path.make_node(os.path.join('Source', 'Android'))

    kw_dict = dict(
        # ...
        android_manifest_path = [ manifest_src_root.abspath() ],
        # ...
    )

    bld.DefineGem(**kw_dict)
```

To learn more about the gem building system and using `wscript` files, see Gem Structure (p. 1073) and
Waf Module Files (wscript) (p. 76). For more information on Android manifests, their purpose, and how to
structure their contents, see the official Android documentation.

**Build and deploy your project for Android**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview.
Download O3DE or visit the AWS Game Tech blog to learn more.

Once you've set up your environment and configured your project, it's time to build your Amazon
Lumberyard project for Android and get it running on a device. In this section, you'll set up an attached
device to use the Shader Compiler, build your project with `lmbr_waf` or Android Studio, and deploy
to a device. You'll also learn how to configure the Lumberyard build system to activate or deactivate
automatically deploying to a device whenever you build.

**Topics**
- Connect to the Shader Compiler (p. 3176)
- Build your project (p. 3177)
- Deployment prerequisites (p. 3177)
- Deploy with Lumberyard Editor (p. 3177)
- Deploy from command line (p. 3178)
Connect to the Shader Compiler

During development, your Lumberyard project connects to the Shader Compiler to generate the shaders for your device. To do this, by set up a reverse proxy between the Android device and the host machine, in order to connect to the Asset Processor. Note that through the VFS you can connect to a remote Asset Processor, which will handle shader compilation. Follow the steps in Serving assets over the Virtual File System (VFS) (p. 3174) to configure VFS.

Important
The Shader Compiler only runs on Windows PCs. If you're using another environment to develop your Android game, you'll need access to a server with the Shader Compiler running and accessible through a firewall. One option is to run a Windows GPU-based EC2 instance. For more information on running a dedicated Shader Compiler service, see Remote Shader Compiler (p. 1645).

To connect to the Shader Compiler

1. Open the lumberyard_install_dir\dev\system_android_es3.cfg file in a text editor.
2. Change the value of r_AssetProcessorShaderCompiler to 1.
   
   ```
   r_AssetProcessorShaderCompiler=1
   ```
3. Change the value of r_ShaderCompilerServer to the IPv4 address of the machine that's running the Shader Compiler. Unless you're running the Shader Compiler in a dedicated environment, use 127.0.0.1.
   
   ```
   r_ShaderCompilerServer=server-ip
   ```

   Important
   Make sure that the IPv4 address of the Android device is in the allow list for your Shader Compiler. Connections to the compiler are made from the device, not proxied through the host.

4. Change the value of r_ShaderCompilerPort to the port that's used by the Shader Compiler. By default, the port for the Shader Compiler is 61453. To see if you're using another port, examine the Remote Shader Compiler Configuration (p. 1646) on the Shader Compiler host.
   
   ```
   r_ShaderCompilerPort=shader-port
   ```
5. Save the file.
6. If running the Shader Compiler locally: Open a command prompt and use adb to redirect traffic at the Shader Compiler port on the device to the Shader Compiler port on your development machine. By default, the port for the Shader Compiler is 61453.
   
   ```
   adb reverse tcp:shader-port tcp:shader-port
   ```

   Important
   This step relies on the adb command being in the PATH environment variable. Follow the instructions in Set up your environment to develop for Android with Lumberyard (p. 3168) to add Android tools to your PATH.

Whether or not you need to manually start the Shader Compiler is dependent upon whether you use the deployment tool. If you're using the Deployment Tool, the Asset Processor automatically starts a Shader
Compiler in the background for you. Otherwise, you need to start the Shader Compiler manually for a deployment. See Running the Remote Shader Compiler (p. 1646) to learn more.

**Build your project**

With asset loading and the shader compiler configured, you're ready to build your Lumberyard project for Android! You can build from the command line, or load the projects created by `lmbr_waf configure` into Android Studio. This section covers how to build your project from the command line, including information on the available Android targets and build system configuration options that are explicitly for use with Android.

**Important**

If you've enabled Incredibuild (p. 118), turn the feature off to be able to compile for Android. Lumberyard doesn't take advantage of the Incredibuild hooks for Android.

**Build with lmbr_waf**

When using `lmbr_waf`, Android builds use the `android_armv8_clang` platform. When building for Android, all of the usual tasks, options, build configurations, and specs are available with `lmbr_waf`. For example, to build the game and engine for Android in profiling mode, use the command `lmbr_waf build_android_armv8_clang_profile -p game_and_engine`.

For more information about the Lumberyard build system, see Using the Waf Build System (p. 63). For information on the Android-specific build settings and `lmbr_waf` command-line options, see the Reference for Android (p. 3187).

**Build with Android Studio**

The Gradle project created by `lmbr_waf configure` can be directly imported into Android Studio without any changes. Follow the instructions from Android Studio - Import Gradle project to import this project, located at `lumberyard_install_dir\dev\Solutions\LumberyardAndroidSDK`.

For further instructions, including how to select targets and configure for debugging, read the Android Studio documentation.

Targets in Android Studio are called *Build Variants* and are named `project-nameLauncher`. Each build variant produces the bootstrap executable that loads the assets and code for the named project, and the Lumberyard engine.

If you're comfortable using the Android build tools from the command line, you can also build any Lumberyard targets using the Android build tools or Gradle. Keep in mind that this only builds your executable and doesn't recompile the assets.

**Deployment prerequisites**

In order to deploy to an Android device, you need the following:

- An Android device that meets the minimum requirements (p. 3167) for Lumberyard. Lumberyard projects won't deploy to an Android emulator.
- The device has developer mode and USB debugging enabled. See Configure developer options in the official Android documentation for instructions on how to enable these features.
- The device is accessible from an Android debug bridge (`adb`) server on the host running the deployment tools.

**Deploy with Lumberyard Editor**

The Lumberyard Editor offers a way to deploy to a mobile device: the Lumberyard Deployment Tool. If you're a content creator who rapidly iterates on their work, this is an ideal way to get on-device testing.
and validation. Using the Deployment Tool requires that you have the Lumberyard Editor and Asset Processor, and an Android device connected to the machine that's running both.

To use the Deployment Tool

1. From the Editor main menu, select File > Project Settings > Deploy to device.
2. In the Deployment Tool UI, select Android ARMv8 for the Target platform value.
3. Select your build configuration, and set the option toggles for Build game and Load current level. It's only necessary to rebuild your game if you're testing updates to the code itself, and not just asset changes.
4. In the Deploy section, select the Local Device tab.
5. Under Deployment Settings, select your target device. Unless you have multiple Android devices connected, there's no need to change this value. Make sure the value for Device IP address is 127.0.0.1. For Android, the Deployment Tool automatically manages port forwarding and connections through the Android debug bridge.
6. Check the Clean device box to remove any files for your project that are copied to the device, including any installed APKs. This is recommended if you're doing a large asset update.
7. Under Asset Options, select your asset mode (p. 3172). Unless you're running the Asset Processor remotely, make sure that the value for Asset Processor IP address is 127.0.0.1 (localhost). If you'll be building shaders through the Shader Compiler, uncheck the Use Asset Processor for shaders checkbox.
8. Select the Deploy to local device button.

Note
If you're using source control for your project, you might see a warning that some files are not writable when you try to deploy the project. Either check these files out in your source control system, or select the Continue button to overwrite them.

Deploy from command line

If you need to deploy less frequently, or can't meet the requirements of the Deployment Tool, deploy-on-build is another option. Deployments during build can be performed automatically, or controlled through a command-line option passed to lmbr_waf. You can also deploy directly from Android Studio if you're using it as your build environment. For more information on using Android Studio for Lumberyard builds, see Build with Android Studio (p. 3177) and the Android Studio documentation.

Important
As part of using deploy-on-build, you'll want to make sure that your assets are properly updated first. This will always involve running the Asset Processor and the shader compiler before a build.

Automatic deploy during build

At the start of development, and for anyone working on the Lumberyard engine or gems that contain code, deploy to a device every time you rebuild. This keeps the APK on the device up to date and ready to test, at a minimal cost of deployment time to the device. As your project grows in size and assets are added, you might choose to turn this feature off – copying assets to the device is what takes the longest amount of time in a deployment.

To modify the build configuration to automatically deploy

1. Open the lumberyard_install_dir\dev\_WAF\_user_settings.options file in a text editor.
2. Edit the [Android Deploy] section of the file to the following:

```plaintext
[Android Deploy]
```
deploy_android = True
deploy_android_clean_device = True
deploy_android_executable = True
deploy_android_replace_apk = True

Command-line options still take precedence over anything in the configuration file. When building for Android and you don't want to deploy, you can add the arguments shown in the next command to keep the current on-device APK installed:

```
lmbr_waf build_android_armv8_clang_profile -p all ^
  --deploy-android-executable=False ^
  --deploy-android-clean-device=False ^
  --deploy-android-replace-apk=False
```

Using this feature with the `deploy_android_armv8_clang_profile` build command is useful when you don't want to replace the executable on the device, but do need to deploy changed assets.

For a full description of all of the available Android deployment options and their effects, see the Reference for Android (p. 3187).

Manual deploy during build

When working from the command line with `lmbr_waf`, you can control deployments by adding arguments. When you're ready to build for deployment, add the arguments `--deploy-android-executable=True` `--deploy-android-clean-device=True` `--deploy-android-replace-apk=True` to your invocation. The following command is used to deploy during a build:

```
lmbr_waf build_android_armv8_clang_profile -p game_and_engine ^
  --deploy-android-executable=True ^
  --deploy-android-clean-device=True ^
  --deploy-android-replace-apk=True
```

During early testing and development, you'll want to always push to a device only when you've confirmed that all of your asset settings, device network configuration, and other build configurations are appropriate for your project. You only need to push new APKs when updating configurations, making code changes, or adding gems.

For a full description of all of the available Android deployment options and their effects, see the Reference for Android (p. 3187).

Deploy with `lmbr_waf`

If your game and engine are already built and up to date, you can run a deployment without doing a build. By running a deploy-only operation, you can update the assets on a device to test and preview them without the cost of a full rebuild. You can even switch the asset deployment type before doing a manual deployment.

Deployments are performed with the `deploy_android_armv8_clang_profile` command. When you're running deployments in this way, make sure to use the `--deploy-android-*` arguments to specify exactly what should be cleaned and which assets or binaries will be deployed to the device. In general, when you're only updating assets, make sure to avoid cleaning assets from the device and don't re-deploy the binaries. The following command shows how to do an asset-only update:

```
lmbr_waf deploy_android_armv8_clang_profile -p all ^
  --deploy-android-executable=False ^
  --deploy-android-clean-device=False ^
```
--deploy-android-replace-apk=False

For a full description of all of the available Android deployment options and their effects, see the Reference for Android (p. 3187).

Next steps

Now that you have your Lumberyard project building and running on an Android device, learn more about mobile-specific design considerations and releasing to the Google Play Store.

- Design Considerations for Creating Mobile Games Using Lumberyard (p. 3214)
- Lumberyard Performance Tuning Guidelines for Mobile Devices (p. 3215)
- Updating Graphics Settings for Android and iOS (p. 3234)
- Releasing Lumberyard projects for Android (p. 3180)

Releasing Lumberyard projects for Android

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Setting up to release for Android can be a complicated process because of the number of steps involved, and the restrictions that are required for publishing on the Google Play Store. For the full and up to date list of requirements, see the Play Store publishing documentation. In this topic you'll walk through additional steps needed for distribution, configure to submit an application larger than 100Mb to the Google Play Store, and build your release binaries.

Topics
- Configure your signing key (p. 3180)
- Create expansion files (OBBs) (p. 3181)
- Release process (p. 3183)

Configure your signing key

As part of submission to the Google Play Store, you’re required to sign your application. The Android signing documentation discusses setting up your key signing in detail, as well as steps involved in uploading your signing key and signed application to Google Play. As part of building your APK, the Lumberyard build system takes care of doing the signing for you.

In order to set up key signing, you’ll need to generate an upload key.

1. Open the file lumberyard_install_dir/dev/_WAF_/android/android_settings.json in a text editor.
2. Make the following edits.
   a. Change the value for DISTRO_KEYSTORE_ALIAS to the alias of your distribution keystore.
   b. Change the value for DISTRO_KEYSTORE to the path to your distribution keystore. This path must be relative to lumberyard_install_dir/dev.
   c. Change BUILD_ENVIRONMENT to Distribution.
"DEV_KEYSTORE_ALIAS" : "development_keystore",
"DEV_KEYSTORE" : "_WAF_/android/dev.keystore",

"DISTRO_KEYSTORE_ALIAS" : "your-keystore-alias",
"DISTRO_KEYSTORE" : "your-keystore-path",

"BUILD_TOOLS_VER" : "latest",
"SDK_VERSION" : "android-28",
"NDK_PLATFORM" : "android-21",

"BUILD_ENVIRONMENT" : "Distribution"
}

Now when building your project, you're required to include the --distro-store-pass and --distro-key-pass arguments. These are respectively the password to the keystore itself and the password to your upload key for the Google Play Store.

This enables your ability to produce release builds. In order to switch back to performing development builds, change BUILD_ENVIRONMENT back to Development.

Create expansion files (OBBs)

Warning
In Lumberyard version v1.21 and later, the resource compiler job which builds OBB files does not ship with the Lumberyard installer. Download the Android_MakeObb job file and place it in the lumberyard_install_dir/dev/Code/Tools/RC/Config/rc directory before continuing.

The MakeObb job uses the old Lumberyard asset management system, where the project-specific resources you include and exclude from the OBB expansions are defined as part of the job.

The Google Play Store has limitations on how applications can be distributed. Android application packages (APKs) are limited to 100MB in size. To ship larger applications, you should use either dynamic content delivery (p. 2155) or Android expansion files. The Lumberyard engine has support for downloading, checking, and extracting expansion files - all you have to do is create them. Google Play allows for uploading two separate OBB files, main and patch, which have a size limit of 2.0Gb each.

How you package your application is an important part of releasing to the Google Play Store. Before continuing in this section, we recommend that you read the Android Developer documentation for APK expansions.

Note
It's important to remember that these restrictions apply only to the Google Play Store. If you're shipping your application to another Android storefront or service that will install it, read the appropriate documentation and understand your distribution channel's requirements.

If you've decided that APK expansions are the right way to distribute your application, follow these steps to configure your project to build OBBs and modify the engine code so that your project will prefer to load loose files extracted from the OBB over searching for assets within pak files.

1. In Windows Command Prompt or Windows Explorer, navigate to the lumberyard_install_dir/dev\Code\Tools\RC\Config\rc folder.
2. Copy the file RCJob_Generic_Android_MakeObb.xml to a new file named RCJob_project-name_Android_MakeObb.xml.
3. Open the file lumberyard_install_dir\dev\project-name\project.json in a text editor.
4. Make the following edits to the file:

   a. If you haven't, set the value of app_obfuscator_salt to a random Base64 string.
b. To enable building main OBB files, set `use_main_obb` to "true".
c. To enable building patch OBB files, set `use_patch_obb` to "true".
d. Set the value of `rc_obb_job` to `RCJob_project-name_Android_MakeObb.xml`.

```
{
    ...
    "android_settings": {
        "package_name": "com.examples.project-name",
        "version_number": 1,
        "version_name": "1.0.0",
        "orientation": "landscape",
        "app_obfuscator_salt": "MY_OBFUSCATOR_SALT_BASE_64",
        "use_main_obb": "(true | false)",
        "use_patch_obb": "(true | false)",
        "rc_obb_job": "RCJob_project-name_Android_MakeObb.xml"
    }
}
```

5. Save and close `project.json`.
6. Open the file `lumberyard_install_dir\dev\Code\Tools\RC\Config\rc\RCJob_project-name_Android_MakeObb.xml` in a text editor.
7. Edit the file to create the OBBs that you wish to build.
   - **If building only a main OBB:** Edit the values of `<Properties>` to properly reference the files that you wish to include and exclude from the main OBB. The default settings include everything in the cache and exclude everything which is pulled into the main APK to launch the engine.
   - **If building only a patch OBB:**
     a. Edit the `zip` attribute on the `PakAssets` and `PakUnpackedAssets` jobs to change references from `${obb_pak}` to `${obb_patch_pak}`.

```
<PakAssets>
  <Job sourceroot="${src_game}" input="${assets_files}"
      zip="${obb_patch_pak}" exclude="${assets_files_excludes}"/>
</PakAssets>

<PakUnpackedAssets>
  <Job sourceroot="${tmp_folder}" input="*." zip="${obb_patch_pak}" exclude=""/>
</PakUnpackedAssets>
```

b. Edit the values of `<Properties>` to properly reference the files that you wish to include and exclude from the patch OBB. The default settings include everything in the cache and exclude everything which is pulled into the main APK to launch the engine.

- **If building both patch and main OBBs:**
  a. Copy the existing `PakAssets` job to a new job, `PakPatchAssets`. Change the values of this new job's attributes as follows.
    - Change the `input` value to `${patch_files}`.
    - Change the `zip` value to `${obb_patch_pak}`.
    - Change the `exclude` value to `${patch_files_excludes}`.

```
<PakPatchAssets>
  <Job sourceroot="${src_game}" input="${patch_files}"
      zip="${obb_patch_pak}" exclude="${patch_files_excludes}"/>
```
b. Add the following new attributes to <Properties>.

- **patch_files** – The files to include in the patch OBB. This **shouldn't** include any files from the APK and should only include files from the main OBB that are being patched. If you're only patching files in your APK, upload a new version to the Google Play Store and require users to update the main distributable instead.

  This value is formatted as a semicolon (;)-separated list of file paths relative to the asset cache. Globbing is supported.

- **patch_files_excludes** – Everything in the cache you wish to exclude from the patch OBB. Unless explicitly including assets from the APK or the main OBB in your patch, this value should include the string #{assets_files};#{assets_files_excludes}.

  This value is formatted as a semicolon (;)-separated list of file paths relative to the asset cache. Globbing is supported.

  **Tip**

  To maintain the default assets_files_excludes list, we suggest renaming it to engine_files_excludes and including this property as part of both your patch_files_excludes and assets_files_excludes.

```xml
<Properties>
  assets_files="main-obb-assets"
  patch_files="patch-obb-assets"

  engine_files_excludes="*cmakelists.*;editor\.*;${levels_pak_excludes};
  ${game_data_pak_excludes};${engine_pak_types};${engine_pak_excludes};
  ${basic_types}"
  assets_files_excludes="assets-not-in-main-obb;${engine_files_excludes}"
  patch_files_excludes="assets-not-in-patch-obb;${engine_files_excludes}"
</Properties>
```

The job to create the APK expansion files is only run when building a **release** configuration. You can always run **debug** or **profile** builds without creating new OBBs.

**Release process**

Now that you've set up your signing key and configured either for dynamic content delivery or set up your OBBs containing assets, it's time to build your release!

1. Run `lmbr_waf configure` to regenerate your Android Studio project. This action updates your project's Android configuration information to indicate that it's building a release for public deployment to a store. You're required to include the --distro-key-pass and --distro-store-pass arguments, now that your project is in Distribution mode.

   ```bash
   lmbr_waf configure --distro-key-pass=key-password --distro-store-pass=store-password
   ```

   **Important**

   You can safely ignore the following warning during configuration for Distribution mode. Your project will still be configured to use the appropriate keystores and take required actions to generate a valid APK for upload to the Google Play Store.
[WARN] The Distribution build environment is not currently supported in Android Studio, falling back to the Development build environment.

If you made any manual changes to your imported Android Studio project, review them to make sure that the project regeneration didn’t change anything important for your release.

2. Disable any remote connections to the Shader Compiler and Asset Processor used in development.
   a. Open the file `lumberyard_install_dir\dev\system_android_es3.cfg` in a text editor.
   b. Change the value of `r_AssetProcessorShaderCompiler` to `0`.

   ```
   r_AssetProcessorShaderCompiler=0
   ```
   c. Open the file `lumberyard_install_dir\dev\bootstrap.cfg` in a text editor.
   d. Change the value of `remote_filesystem` to `0`.

   ```
   remote_filesystem=0
   ```
   e. Change the value of `connect_to_remote` to `0`.

   ```
   connect_to_remote=0
   ```
   f. Save and close the file.

3. Make sure that the Shader Compiler is running, at the same network address and port used for building shaders during development. Unlike development builds, release uses the `lumberyard_install_dir\dev\shadercachegen.cfg` configuration file to connect to the Shader Compiler during release builds.
   a. Open `lumberyard_install_dir\dev\shadercachegen.cfg` in a text editor.
   b. Edit the value of `r_ShaderCompilerServer` to set it to the IPv4 address of the shader server used in development.
   c. When using the shader compiler on localhost (127.0.0.1): Start the Shader Compiler server on your machine by launching it from `lumberyard_install_dir\dev\Tools\CrySCompileServer\x64\profile\CrySCompileServer.exe`.

4. With everything configured, it’s time to build your release. Build with `lmbr_waf` and supply your distribution key information on the command line.

   ```
   lmbr_waf build_android_armv8_clang_release ^
   --distro-store-pass=distribution-keystore-password ^
   --distro-key-pass=distribution-key-password ^
   -p game_and_engine
   ```

Now your APK and any configured OBBs are ready for testing and upload to the Google Play Store! Your build products are located in the following places.

- **OBB files**: Located in `lumberyard_install_dir\dev`
- **APK**: `lumberyard_install_dir\dev\BinAndroidArmv8Clang.Release\project-nameLauncher_w_assets.apk`

If you inspect the contents of the APK, you’ll see that the only assets included are those needed for configuration and launching of the engine.
Troubleshooting Lumberyard issues on Android

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Things go wrong during development, and knowing which issues are the most commonly encountered and how to resolve them can be a big help. This topic is a knowledge base of many of the issues that users experience when they’re working with Amazon Lumberyard to develop a project for Android. This section focuses on helping you quickly identify your issue and get it resolved, so that you can get back to work.

For information on how to design assets and configure renderer settings to improve performance, see Lumberyard Performance Tuning Guidelines for Mobile Devices (p. 3215).

All of the advice in this section is specific to issues you’ll encounter when building with the Lumberyard tools, or when you’re running a project on a device and see obvious errors caused by the engine itself. For general advice and instructions on debugging your Lumberyard project for Android, see the Android Studio Debugger documentation.

If you have a question that isn’t answered here, try checking in with the Lumberyard community using one of these resources:

- The Lumberyard forums
- The unofficial Lumberyard Discord server

Topics
- Black screen when launching on a device (p. 3185)
- Maven repository warnings (p. 3186)
- Debug release APKs (p. 3187)

Black screen when launching on a device

Oftentimes, launching a build and seeing a black screen means that either a map failed to load, or shaders are missing. You can diagnose and resolve the problem by trying the following actions:

- Inspect logs from adb. Log messages from the Lumberyard engine include the string LMBR, and you can filter the output from adb logcat to display only these messages. Use the following command in a Windows command prompt to get the current device logs:

  ```
  adb logcat -s LMBR
  ```

  For more information on adb logcat and how to improve filtering and which messages are displayed, see the official Android logcat command-line tool documentation.

- Make sure that a map is being loaded upon the application launch by following these steps:
  1. Open the file lumberyard_install_dir/dev/project-name/autoexec.cfg in a text editor.
  2. Check the file for a map command. If the map is missing, or references an incorrect map name, add the following line or change the map value:

    ```
    map level-to-load
    ```
• Check to make sure that the device is allowed to connect to the shader compiler. Run through the following steps to check your configuration and make sure that everything is correctly enabled:

1. If the Shader Compiler is on a remote host, make sure that inbound traffic to the shader compiler's port on the host is allowed. By default, this port is 61453. See also Remote Shader Compiler (p. 1645) for full information on configuring to connect to a remote shader compiler.

2. If the Shader Compiler is running on the same machine with the Android device connected to it, make sure that port forwarding is set up to send traffic to the host from the Android device. By default, the Shader Compiler's port is 61453. Configure device port forwarding by running the following command in a Windows command prompt.

   ```
   adb reverse tcp:shader-compiler-port tcp:shader-compiler-port
   ```

3. Close the Shader Compiler, delete any produced assets, and restart it. Shader Compiler assets for Android are placed in `lumberyard_install_dir\dev\Cache\project-name\es3\user\cache\shaders`.

4. Delete the application from your device, rebuild, and re-deploy.

To run through the full process of setting up a connection from your device to the Shader Compiler, see Connect to the Shader Compiler (p. 3176) or Serving assets over the Virtual File System (VFS) (p. 3174).

### Maven repository warnings

As part of a Lumberyard build or configure, you might see some warnings similar to the following.

```
[WARN] Failed to connect to https://maven.google.com/androidx/compose/ui/group-index.xml. Access to Google's main Maven repository may be incomplete.
[WARN] Failed to connect to https://maven.google.com/androidx/core/group-index.xml. Access to Google's main Maven repository may be incomplete.
```

Behind the scenes, `lmbr_waf` is invoking the Android build tools, which use the Gradle build system. Gradle pulls up-to-date libraries and does some versioning dependency checks by pulling information from Google's Maven repositories. These warnings aren't fatal and shouldn't prevent you from producing a build, but you can remove them by updating the Gradle version for the Android Studio project generated by `lmbr_waf configure`.

To remove Maven repository errors

1. Import the project located at `lumberyard_install_dir\dev\Solutions/LumberyardAndroidSDK` into Android Studio. See Import a Gradle project to Android Studio.

2. Wait for the project to load and for the initial Gradle configure to complete.

3. At a certain point during the configuration, Android Studio will report that the Gradle plugin is ready to update by displaying a popup notification. Select update in this notification.

   ![Plugin Update Recommended](image)

   **Android Gradle Plugin is ready to update.**

4. Another dialog box will appear, informing you of the version to update to and giving you the option to see release notes. Select the **Update** button in this dialog to update the Gradle plugin and remove build warnings.
Note
If you ever regenerate your Android Studio project, you'll need to perform these steps again. As long as the project isn't overwritten, you can safely run `lmbr_waf configure` without needing to make more changes.

Debug release APKs

As part of building for release, debugging symbols are stripped from the final executable. Whether you're trying to diagnose issues before release or working on something reported by a user in your Google Play Store release, the solution is the same. Android uses special metadata associated with the APK to determine whether or not the Android debugger is allowed to connect to a release binary.

This setting is controlled through the Gradle build system. See the Android Studio documentation on enabling debugging for build variants, and debugging pre-build APKs. This documentation has a full set of instructions on how to use debug symbols generated by earlier builds with release APKs.

Reference for Android

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

This section is a reference for commands and build settings used by Lumberyard when working with the Android platform. For general information and system-wide build settings or commands, see the following topics.

- Waf Commands and Options (p. 90)
- Waf Project Settings (p. 96)

Topics

- Build command reference (p. 3187)
- Waf settings (p. 3188)
- Android build configuration file (p. 3190)
- Other command-line arguments (p. 3191)
- Lumberyard project settings for Android (p. 3191)

Build command reference

This section contains information on the available build variants and commands that are used to build and deploy Android projects through `lmbr_waf`. The `lmbr_waf` commands are structured as `command_android_armv8_clang_variant`. For example, you deploy a debug build to a device with the following command:

```
lmbr_waf deploy_android_armv8_clang_debug -p game_and_engine
```

Tip
Since you never need to build the editor or any tools for Android, you can always use the `game_and_engine` project spec with any Android commands.

Valid values for all of these parts of a command are indexed here. You can also display all available Android build commands by using `lmbr_waf --help` from the command line.
**Commands**

**build**
Build the currently selected Lumberyard project.

**deploy**
Manually deploy to a connected device.

**clean**
Clean local build artifacts. This command doesn't perform any device cleanup.

**package**
Package the contents of your project into a single distributable APK.

**Build variants**

**debug**
Enables debugging. Does not change or disable any optimizations during compile time.

**profile**
Enables debugging. Does not change or disable any optimizations during compile time.

**release**
Disables debugging and strips debug symbols in the final build.

**Waf settings**

The Waf build system settings can be controlled by either setting an attribute value in `lumberyard_install_dir/dev/_WAF_/user_settings.options`, or by passing an argument on the command line. The configuration file is written in INI file format. Each of the following reference tables are named after their corresponding INI section header.

For a full reference of all configurable Waf settings, see Waf User Options and Settings (p. 78).

**Android Options**

<table>
<thead>
<tr>
<th>Attribute (config file)</th>
<th>Parameter (command line)</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>android_maven_force_http</td>
<td>--android-maven-force-http-requests</td>
<td>Forces Android Maven library requests to use HTTP instead of HTTPS.</td>
<td>False</td>
</tr>
<tr>
<td>android_asset_mode</td>
<td>--android-asset-mode</td>
<td>Specify one of the following asset packaging modes:</td>
<td>configuration_default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• configuration_default – Use the current build configuration to determine how to package the assets (for example, loose_files for debug or profile, project_settings for release or performance).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• loose_files – No additional processing is done on the compiled assets.</td>
<td></td>
</tr>
</tbody>
</table>
### Android Deploy

<table>
<thead>
<tr>
<th>Attribute (config file)</th>
<th>Parameter (command line)</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>apk_files</code> – The compiled assets are packaged inside the APK.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>project_settings</code> – Use <code>project.json</code> settings to determine the asset packaging mode.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information on asset modes, see [Handling assets on Android](p. 3172).</td>
<td></td>
</tr>
</tbody>
</table>

**Android Deploy**

<table>
<thead>
<tr>
<th>Attribute (config file)</th>
<th>Parameter (command line)</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><code>deploy_android</code></td>
<td>True</td>
</tr>
<tr>
<td></td>
<td><code>deploy-android</code></td>
<td>Deploys to an Android device after a successful <code>package</code> command.</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td><code>deploy-android-clean-device</code></td>
<td>Completely uninstalls the app from the target device.</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td><code>deploy-android-device-filter</code></td>
<td>Comma-separated list of Android device IDs to deploy to. If empty, deploys to all connected devices.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>deploy-android-executable</code></td>
<td>Installs the executable <code>.apk</code> file on the Android device.</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td><code>deploy-android-install-options</code></td>
<td>Provides additional options to specify for the <code>install</code> command.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>deploy-android-apk</code></td>
<td>When installing the <code>.apk</code> file to the Android device, uses the <code>-r</code> option to force the replacement of the package.</td>
<td>True</td>
</tr>
</tbody>
</table>

**Android Project Generator**

<table>
<thead>
<tr>
<th>Attribute (config file)</th>
<th>Parameter (command line)</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>android_studio_project_folder</code></td>
<td>Solutions</td>
</tr>
<tr>
<td></td>
<td><code>android_studio_project_folder</code></td>
<td>Name of the directory in which the generated Android Studio project should be stored.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>android_studio_project_name</code></td>
<td>Name of the root Android project directory and Android Studio project name.</td>
<td>LumberyardAndroidSDK</td>
</tr>
</tbody>
</table>
Android build configuration file

Lumberyard stores some settings for the Android platform in an independent file, `lumberyard_install_dir\dev\_WAF\android\android_settings.json`. This file contains a JSON dictionary of build setting names and their values.

**Important**

After making any changes to the Android settings, run `lmbr_waf configure`. Taking this action causes your changes to be imported to the generated Android Studio project and its manifest information.

### DEV_KEYSTORE_ALIAS

The keystore alias used in the Development environment by the Android JAR signer. This value is exported to the build command's environment as `KEYSTORE_ALIAS`.

### DEV_KEYSTORE

The path to the keystore used in the Development environment by the Android JAR signer. This path should either be absolute or relative to the `lumberyard_install_dir\dev` directory. This value is exported to the build command's environment as `KEYSTORE`.

### DISTRO_KEYSTORE_ALIAS

The keystore alias used in the Distribution environment by the Android JAR signer. This value is exported to the build command's environment as `KEYSTORE_ALIAS`.

### DISTRO_KEYSTORE

The path to the keystore used in the Distribution environment by the Android JAR signer. This path should either be absolute or relative to the `lumberyard_install_dir\dev` directory. This value is exported to the build command's environment as `KEYSTORE`.

### BUILD_TOOLS_VER

The version of the Android build tools to use. The value can either be `latest` to use the most recent version of the Android build tools, or a specific version identifier.

### SDK_VERSION

The SDK version to use. The value must be in the format `android-apilevel`, where `apilevel` is the API level of the SDK.

### NDK_PLATFORM

The NDK version to use. The value must be in the format `android-version`, where `version` is the major version number of the NDK.

### BUILD_ENVIRONMENT

Selects which environment is being built for. Valid values are Development and Distribution (case-sensitive).

#### Development

The default environment, used for daily development. Building for the Development environment doesn't require a key store or signing key.
Distribution

Build for distribution to the Google Play Store. This environment requires that you use the --distro-key-pass and --distro-store-pass arguments to access the distribution key store and your signing key.

Other command-line arguments

Android builds use some command-line arguments that are not stored in a configuration file because they would contain sensitive data, or are best for one-off use. There are also some arguments that override some settings that are less commonly changed.

--android-sdk-version-override

Override the Android SDK version used in the Java compilation. Only works during configure.

--android-ndk-platform-override

Override the Android NDK platform version used in the native compilation. Only works during configure.

--dev-store-pass

The password for the development keystore. Lumberyard ships with a store with the password Lumberyard by default. This default password is used in the Development environment if this command-line argument is missing.

--dev-key-pass

The password for the development key. Lumberyard ships with a self-signed key with the password Lumberyard in its default key store. This default password is used in the Development environment if this command-line argument is missing.

--distro-store-pass

The password for the distribution keystore. This argument is required when using the Distribution environment.

--distro-key-pass

The password for the distribution key. This argument is required when using the Distribution environment.

--deploy-android-attempt-libs-only

Push only the changed native libraries. If deploy_android_executable is enabled, it takes precedence if the executable was modified. This option is ignored if deploy_android_clean_device is enabled. This argument only works when deploying to a rooted device.

Lumberyard project settings for Android

As part of every Lumberyard project, there is a settings file that describes the application and per-project build settings. This file is located at lumberyard_install_dir/dev/project-name/project.json. Within the JSON dictionary contained in this file is the key android_settings, with a value of a JSON dictionary that contains the settings used by your project when building and deploying for Android.

package_name

Android application package identifier. Used for generating the project-specific Java activity class and as part of AndroidManifest.xml. The identifier must be in reverse domain name format.

For more information, see the Android Application ID documentation.
Type: String

Default: "com.lumberyard.sdk"

**version_number**

Internal application version number. Used to set the "android:versionCode" value of AndroidManifest.xml.

Type: Integer (32-bit)

Default: 1

**version_name**

Human-readable version number, displayed in the Google Play Store. Used to set the "android:versionName" value of AndroidManifest.xml.

Type: String

Default: "1.0.0.0"

**orientation**

The orientation of the Android app. Used to set the "android:screenOrientation" value of AndroidManifest.xml.

For valid values, see the Android Activity documentation.

Type: String

Default: "landscape"

**icons**

A map of icon override paths based on screen DPI. All entries should be paths relative to
lumberyard_install_dir/Code/project-name/Resources or an absolute path.

Available options:
- default – Default image used if a specific DPI is not specified
- mdpi – Medium ~160 DPI
- hdpi – High ~240 DPI
- xhdpi – Extra high ~320 DPI
- xxhdpi – Extra-extra high ~480 DPI
- xxxhdpi – Extra-extra-extra high ~640 DPI

For more information on Android screen DPI settings, see the Android screen compatibility documentation.

**Example**

```json
"icons" : {
  "default" : "AndroidLauncher/icon-xhdpi.png",
  "mdpi" : "AndroidLauncher/icon-mdpi.png",
  "hdpi" : "AndroidLauncher/icon-hdpi.png",
  "xhdpi" : "AndroidLauncher/icon-xhdpi.png",
  "xxhdpi" : "AndroidLauncher/icon-xxhdpi.png",
  "xxxhdpi" : "AndroidLauncher/icon-xxxhdpi.png"
}
```
app_public_key

The app license key that the Google Play store provides. Required for using APK expansion files or other Google Play Services.

Type: String

Default: "NoKey"

app_obfuscator_salt

Salt used in the signing of OBB files.

Type: String (Base64)

Default: ""

use_main_obb

Specify if the "Main" APK Expansion file should be used. This option toggles the APK expansion file mode in release builds.

Type: String ("true" | "false")

Default: "false"

use_patch_obb

Specify if the "Patch" APK expansion file should be used. This option toggles the APK expansion file mode in release builds.

Type: String ("true" | "false")

Default: "false"

enable_key_screen_on

Enable or disable the screen wake lock. If true, the device won't go to sleep while the activity is active.

Type: String ("true" | "false")

Default: "false"

disable_immersive_mode

Disable the hiding of top and bottom system bars.

Type: String ("true" | "false")

Default: "false"

rc_pak_job

Path to the RC job XML file that's used to override the normal APK expansion file generation that's used in release builds.

The path must be relative to lumberyard_install_dir/dev/Bin64/rc.

Type: String (Path)

Default: "RCJob_Generic_Android_MakeObb.xml"

multi_window_options

Information for multi-window support when targeting Android N (API 24) or later.
The keys used in this dictionary for window settings are as follows:

**enabled**

Activate or deactivate multi-window support.

Type: Boolean

Default: false

**default_width**

The preferred density-independent pixel width of the application when launching in multi-window mode. This measurement is in dp, which is always calculated against a 160dpi screen (1dp = 1 pixel on a 160dpi screen).

Type: Integer

Default: None

**default_height**

The preferred density-independent pixel height of the application when launching in multi-window mode. This measurement is in dp, which is always calculated against a 160dpi screen (1dp = 1 pixel on a 160dpi screen).

Type: Integer

Default: None

**min_width**

The minimum density-independent pixel width that your application can run in multi-window mode. Your application is cropped if the window is smaller than this value.

This measurement is in dp, which is always calculated against a 160dpi screen (1dp = 1 pixel on a 160dpi screen).

Type: Integer

Default: None

**min_height**

The minimum density-independent pixel height that your application can run in multi-window mode. Your application will be cropped if the window is smaller than this value.

This measurement is in dp, which is always calculated against a 160dpi screen (1dp = 1 pixel on a 160dpi screen).

Type: Integer

Default: None

**gravity**

The preferred starting location of the window when launched in multi-window mode.

See the Android LinearLayout documentation for valid values.

Type: String

Default: None

Type: Dictionary

Default: Empty
iOS Support

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Lumberyard to create applications for iOS devices (p. 3166). The topics in this section will teach you how to generate assets that your app can load, and how to build and debug code. You will also learn about the different types of apps that you can generate for development purposes or to release publicly.

Lumberyard includes four iOS-supported sample projects that you can use to learn how to build assets for iOS games using Asset Processor, build shaders using the remote shader compiler, and build and deploy iOS apps using the Lumberyard build tools. For more information, see Using Lumberyard sample projects and levels (p. 142).

Lumberyard has the following minimum requirements for iOS:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>ARM v8 or newer</td>
</tr>
<tr>
<td>GPU</td>
<td>A8 or newer</td>
</tr>
<tr>
<td></td>
<td>Supports Metal</td>
</tr>
<tr>
<td>OS</td>
<td>iOS 13 or newer</td>
</tr>
<tr>
<td>Example devices</td>
<td>iPhone 6 and newer</td>
</tr>
</tbody>
</table>

Prerequisites

To build apps for iOS devices, you must have the following:

- Xcode 11 or later
- iOS 13 SDK or later
- Lumberyard Mac Support Files
- Lumberyard installed on a Windows computer
- Lumberyard installed on a macOS computer
- Familiarity with Lumberyard Editor, the Shader Compiler, and Asset Processor
- Ability to build from a command line tool
- Previously built the macOS code and optionally the PC code

To set up your environment, use the Lumberyard Setup Assistant (located in the lumberyard_version\dev\Tools\LmbrSetup\Mac directory) to select the following:

- Create, Modify and Build projects
- Compile for iOS devices

The Setup Assistant will install the required third-party software and SDKs.

Lumberyard Binaries

During installation, Lumberyard creates the following directories for its binaries:
Mac

- BinMac64

PC

- Bin64vc142 (for binaries generated with Visual Studio 2019)

Anatomy of a Lumberyard iOS App

The Lumberyard build system can generate the following types of applications:

- Development app
- VFS app
- Release app

**Note**

Lumberyard Editor is currently supported on Windows only. You must export each level of your game project prior to using Asset Processor for your assets.

Development App

You will use a Development app throughout most of your development cycle. Your app must connect to the Remote Shader Compiler in order to compile the shaders into a format that the iOS device can read. Your app connects to the shader compiler over WiFi or through Asset Processor. Asset Processor generates the Development app executable and its assets, which can be loose files or PAK files.
You can use the VFS app with the Virtual Filing System (VFS). While similar to a Development app, the VFS app also contains the `bootstrap.cfg` and `game.xml` configuration files in your build. After reading these configuration files, the app connects to your macOS or Windows computer and uses the assets from the local Cache directory. This allows you to iterate content and see the changes on your device in real time.

Your app must also connect to the Remote Shader Compiler in order to compile the shaders into a format that the iOS device can read. Your app connects to the shader compiler over WiFi or through Asset Processor on startup. You can optionally choose to direct traffic to and from the Remote Compiler through Asset Processor.
Release App

The Release app contains the executable and all binaries and assets that are required for your app to run. The assets and shaders must be PAK files because the app will not connect to the Remote Shader Compiler or Asset Processor.
Topics

- Quick Start: Running the Samples Project on iOS Devices (p. 3200)
- Building Your iOS App (p. 3201)
- Creating a Release App (p. 3204)
- Using Virtual File System with iOS (p. 3208)
- Universal Remote Console (p. 3209)
- Sharing Assets Between Windows and macOS (p. 3211)
- Add iOS Frameworks to a Lumberyard project (p. 3212)
- iOS Debugging and Troubleshooting (p. 3213)
Quick Start: Running the Samples Project on iOS Devices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use this quick start to learn how to run the Samples Project (p. 146) on your iOS device from a macOS computer.

This topic assumes that you have:
- Xcode installed on your macOS computer
- Lumberyard installed on your macOS computer
- An iOS device connected to your computer with a USB cable
- Lumberyard installed on a Windows computer
- Remote Shader Compiler running on your Windows or macOS computer

For more information about creating iOS apps, see iOS Support (p. 3195).

To run the Samples Project on an iOS device
1. Use Lumberyard Setup Assistant to install the required third-party software and SDKs for iOS.
   a. On your macOS computer, navigate to the `lumberyard_version\dev\Tools\LmbrSetup\Mac` directory and double-click `SetupAssistant.exe`.
   b. If you see a Custom Install box, click Customize and then verify that the engine root path is correct.
   c. On the Get Started page, select the following:
      - Create, Modify and Build projects
      - Compile for iOS devices
   d. Complete the Lumberyard Setup Assistant wizard. For more information, see Running Lumberyard Setup Assistant (p. 16).
2. Configure Asset Processor for iOS.
   a. Use your preferred text editor to open the `AssetProcessorPlatformConfig.ini` file. You can find this file in the `lumberyard_version\dev\` directory.
   b. Remove the preceding semicolon to uncomment `ios=enabled`.
   c. Save the file.
3. Launch Asset Processor.
   a. Navigate to the following directory and double-click `AssetProcessor.exe`:
      - On Windows: `lumberyard_version\dev\Bin64vc141\`
      - On macOS: `lumberyard_version\dev\BinMac64`
   b. In Asset Processor, verify that the Status is Idle.
4. In a command line or Terminal window, do the following:
   a. Change directory to the `lumberyard_version\dev\` directory.
   b. Type the following:
      • On Windows: `lmbr_waf.bat configure`
      • On macOS: `sh lmbr_waf.sh configure`

5. On your Windows or macOS computer, set the IP address for the remote shader compiler server.
   a. Use your preferred text editor to open the `system_ios_ios.cfg` file. You can find this file in the `lumberyard_version\dev\` directory.
   b. Set `r_ShaderCompilerServer` to the IP address of your Windows computer that runs the shader compiler.
   c. Save the file.

6. Build and deploy the code.
   a. Navigate to the `lumberyard_version\dev\Solutions\` directory and open the `LumberyardiOSSDK.xcodeproj` file.
   b. Click **Run** to build and deploy the code to your iOS device.

**Building Your iOS App**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Set up your build environment and build an app that can run on an iOS device.

You must do the following:

1. Configure a game project (p. 3201).
2. Prepare your assets (p. 3202).
3. Configure the build system (p. 3202).
4. Modify your user settings (p. 3203).
5. Run the Remote Shader Compiler (p. 3203).
6. Build code from the command line (p. 3204).

**Step 1: Configure a Game Project**

You have two options for configuring a game project:

- Use the **Samples Project** (p. 142) in the `lumberyard_version\dev\` directory as a foundation for your own game project.
- Create a new game project. For more information, see Creating and Switching Game Projects (p. 44).
To configure a game project

1. On your Windows computer, navigate to the \lumberyard_version\dev\Tools\LmbrSetup\Mac\ directory and open the Project Configurator.
2. In the Project Configurator, do one of the following:
   - Select SamplesProject and then click Set as default. Follow the instructions for switching game projects (p. 48).
   - Click Create new. Follow the instructions for creating a game project (p. 44).
3. If you created a new game project, do the following:
   - a. In the Project Configurator, select your game project name and then click Set as default.
   - b. In Lumberyard Editor, open your level and then press Ctrl+E. This will export the levels you have created.
4. (Optional) Configure system components and memory settings for your game project. For more information, see Configuring Advanced Settings (p. 49).

Step 2: Prepare Your Assets

You can generate the data for your app on a computer running either Windows or macOS. If you choose to use a Windows computer, you must configure your Windows computer and iOS device to share the same network. For more information, see the Run the Remote Shader Compiler (p. 3203) page.

To prepare your assets

1. Do the following to configure Asset Processor for iOS:
   - a. Navigate to the \lumberyard_version\dev\ directory.
   - b. Use your preferred text editor to open the AssetProcessorPlatformConfig.ini file.
   - c. Remove the preceding semicolon to uncomment ios=enabled.

   ```
   [[Platforms]
   ;pc=enabled
   ;es3=enabled
   ios=enabled
   ;osx_gl=enabled
   ```
   - d. Save the file.
2. Do the following to launch Asset Processor:
   - a. Navigate to the following directory:
     - On a Windows computer: \lumberyard_version\dev\Bin64vc141
     - On a macOS computer: \lumberyard_version\dev\Bin64Mac/
   - b. Double-click AssetProcessor.exe.
   - c. In Asset Processor, verify that the Status is Idle.

Step 3: Configure the Build System

When you build the engine and tools code for macOS or Windows, you initialize the build system and generate project files for Xcode.

To configure the build system

1. In a command line window, type sh \lmbr_waf.sh configure
2. Verify that you see the following in the comments from the build system:

```
mac config
[WAF] 'xcode_ios' finished successfully (3.610s)
[WAF] 'xcode_mac' finished successfully (3.507s)
```

### Step 4: Modify Your User Settings

Update your configuration file to help increase the compilation speed and deployment when you build your game project.

**To modify your user settings**

1. Use your preferred text editor to open the `user_settings.options` file. You can find this file in the `lumberyard_version\dev\WAF\` directory.
2. Under `[Build Options]`, set `use_uber_files` to `True`.
3. Save the file.

### Step 5: Run the Remote Shader Compiler

Lumberyard uses a versatile shader system to achieve high quality, realistic graphics. When you run a game on an iOS device during development, you must connect to a remote shader compiler on your Windows or macOS computer. This compiles the subset of shaders required by your game, on demand.

When a new shader is compiled, the game waits for the remote shader compiler to compile the binary shader permutation and then send it back to your device. Once this occurs, the shader is cached on your device until you delete the app. When you are ready to release your game, you must pack up and include all cached binary shaders.

**Note**

You can also run the remote shader compiler on an Amazon EC2 instance. For information, see [Running the Shader Compiler on Amazon EC2](p. 3239).

**Prerequisites**

To use the remote shader compiler, you must do the following:

- (First time only) Create a `config.ini` file that tells the remote shader compiler the addresses from which to accept connections. For instructions, see the procedure below.
- Connect the remote shader compiler host computer and iOS device to the same network and configure any firewalls to allow traffic through port 61453.
- Set up the system configuration file (`system_ios_ios.cfg`) for the mobile device to connect to the remote shader compiler on your computer.

You can use an allow list to specify the IP addresses that are allowed to connect to your remote shader compiler. For information, see [Creating an allow list for the Remote Shader Compiler](p. 1647).

**Enabling a Connection Between the iOS App and the Remote Shader Compiler**

You must modify certain configuration files to allow your iOS app to connect to the shader compiler.

**To allow your iOS app to connect to the shader compiler**

1. Do the following on your macOS computer:
   
   a. Use your preferred text editor to open the `system_ios_ios.cfg` file. You can find this file in the `lumberyard_version/dev/` directory.
b. Set the r_ShaderCompilerServer console variable to the IP address of the host computer on which you are running the shader compiler.

c. Save the file.

2. Do the following on your computer that runs the shader compiler:

a. If one doesn't yet exist, create a config.ini file in one of the following directories:
   - On Windows: `lumberyard_version\dev\Tools\CrySCompileServer\x64\profile\`
   - On macOS: `lumberyard_version/dev/Tools/CrySCompileServer/osx/profile/`

b. Use your preferred text editor to open the config.ini file.

c. For whitelist= <device_ip_address>, replace <device_ip_address> with the IP address, in CIDR format, of your iOS device.

d. Save the file.

3. On your shader compiler host computer, launch CrySCompileServer from one of the following directories:

   - On Windows: `lumberyard_version\dev\Tools\CrySCompileServer\x64\profile\`
   - On macOS: `lumberyard_version/dev/Tools/CrySCompileServer/osx/profile/`

**Step 6: Build Code from a Command Line**

Configure and build various targets of your app, depending on your mode of development. You can create a debug, profile, or release build.

- **Debug** – Allows you to see your code running under a debugger. This build is slowest to run.
- **Profile** – Allows you to debug your code, though some code may be optimized and difficult to trace. This build runs faster on your iOS device.
- **Release** – Includes all required asset and shader .pak files for a release version of your iOS game. This build runs the fastest; however, special steps are required to generate the build. For more information, see Creating a Release App (p. 3204).

When you build code from a command line, you must take an additional step to run your app on an iOS device. You use Xcode to open the LumberyardiOSSDK.xcodeproj project that is generated in the `lumberyard_version/dev/Solutions/` directory. Then you select your device, follow the standard procedure to build and run on the device, set breakpoints, and inspect variables.

**To build code from a command line**

1. On your macOS computer, in a Terminal window, navigate to the `lumberyard_version/dev/` directory.

2. Build various targets of your game:

   - To build debug, type: `sh lmbr_waf.sh build_ios_debug -p all`
   - To build profile, type: `sh lmbr_waf.sh build_ios_profile -p all`
   - To build release, type: `sh lmbr_waf.sh build_ios_release -p all`

**Creating a Release App**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Once you have finished your Lumberyard iOS game, you can prepare it for the App Store by creating a release app and including the assets and shaders in .pak files. You must do the following:

1. **Prepare your code (p. 3205).**
2. **Modify your settings (p. 3205).**
3. **Create .pak files (p. 3205).**
4. **Configure the App Store settings (p. 3207).**
5. **Generate a build (p. 3207).**

## Preparing Your Code

Modify the code in the IConsole.h file to enable the creation of mobile release builds.

### To modify the IConsole.h file

1. On your macOS computer, navigate to the `lumberyard_version\dev\Code\CryEngine\CryCommon\` directory.
2. Use your preferred text editor to open the `IConsole.h` file.
3. Modify line 41 as follows:

   ```cpp
   //Enable modification of CVARS for mobile release builds
   #if defined(AZ_PLATFORM_APPLE_IOS)
   #define ALLOW_CONST_CVAR_MODIFICATIONS 1
   #else
   #define ALLOW_CONST_CVAR_MODIFICATIONS 0
   #endif
   ```

4. Save the file.

## Modifying Your Settings

Modify the settings in your system configuration file to disable communication between the iOS device and the remote shader compiler. For release builds, Lumberyard packages the shaders directly to the iOS device instead of compiling shaders on demand.

### To modify the system configuration file

1. On your macOS computer, navigate to the `lumberyard_version\dev\` directory.
2. Use your preferred text editor to open the `system_ios_ios.cfg` file.
3. Set the following console variables to 0.

   ```
   • r_AssetProcessorShaderCompiler
   • r_ShadersRemoteCompiler
   • r_ShadersAllowCompilation
   ```

4. Save the file.

## Creating .Pak Files

Lumberyard apps use .pak files to include assets and shaders for your app build. The .pak files are required for release apps and must be created manually.

### Topics

- Building Shader .Pak Files (p. 3206)
- Deploying Shader .Pak Files (p. 3206)
Building Shader .Pak Files

Use the Remote Shader Compiler to generate the shaders that are packed into your app build. You must run your app in profile or debug and view every surface in your game level to capture all shader permutations. Lumberyard supports both GMEM 128 and GMEM 256 and generates shaders on the highest version that a device supports. As a result, you must run your game on two devices, one that supports GMEM 128 and one that supports GMEM 256.

To generate shaders for your iOS app

1. Build, deploy, and run your game in profile or debug mode. For information, see Building Your iOS App (p. 3201).
2. In your game, explore every area in every level to ensure that all shader permutations required for the game are generated. Exit the game when you are finished.

To build a shader .pak file

1. Run the Remote Shader Compiler. For more information, see Running the Remote Shader Compiler (p. 1646).
2. In a command line window, navigate to the \lumberyard_version\dev\ directory and locate the lmbr_pak_shaders.sh file.
3. To use the lmbr_pak_shaders.sh file, enter a command that provides the name of the game project and the platform.

```
lmbr_pak_shaders.sh game_project_name METAL ios
```

Example

To build the shaders for the Samples Project, enter the following command:

```
lmbr_pak_shaders.sh SamplesProject METAL ios
```

Deploying Shader .Pak Files

When the batch file finishes building the shader .pak file for your game project, verify that the following files exist in the \lumberyard_version\dev\Build\ios\game_project_name\ directory:

- ShaderCache.pak – Contains all compiled shaders that are used only when the shader cannot be found in the current level's shader cache.
- ShaderCacheStartup.pak – Contains a subset of compiled shaders that are used to accelerate the startup time of the engine.

To pack assets for your app

1. Navigate to the \lumberyard_version\dev\ directory.
2. Copy the Build<game_project_name>_Paks_PC.bat file and rename it to Build<game_project_name>_Paks_iOS.bat.
3. Use your preferred text editor to open the Build<game_project_name>_Paks_iOS.bat file.
4. Edit the file as follows:
   - Line 20: \%BINFOLDER%\AssetProcessorBatch.exe /gamefolder=game_project_name /platforms=ios
iOS Support

- Line 25: `%BINFOLDER%\rc\rc.exe /job=%BINFOLDER%\rc
\RCJob_Generic_MakePaks.xml /p=ios /game=game_project_name

5. Save the file.
6. Run the `Build<game_project_name>_Paks_iOS.bat` file and wait for the .pak files to finish building.

Using Shader .Pak Files

Add the shader .pak files to your Xcode project.

To use shader .pak files

1. Navigate to the `lumberyard_version\dev\Cache\game_project_name\ios_paks\` directory. The generated .pak files are saved here.
2. Do one of the following to add the .pak files to your Xcode project:
   - If you're sharing the Cache directory between your Windows and macOS computers, reopen your Xcode project. This will add the files to your Resources directory.
   - Copy the `ios_paks` directory and save it in the `lumberyard_version\dev\Cache\game_project_name\` directory on your macOS computer. Then reopen your Xcode project.

Configuring the App Store Settings

Every Lumberyard project includes an Info.plist file that contains the default values for the following settings:

- Display name
- App icon
- Splash screen
- Screen orientation
- Other related settings

You can access the Info.plist file in the `lumberyard_version\dev\Code\game_project_name\Resources\IOSLauncher\` directory. The default app icons and splash screens are in the Images.xcassets directory. For information about using the default app icons and splash screens, see the Lumberyard Logos and Branding Guidelines.

Verify the Info.plist file is writeable and then use Xcode to modify the values for these settings. For information about modifying these settings in Xcode, see Configuring Your Xcode Project for Distribution.

Generating a Build

After generating all required asset and shader .pak files, you can use Xcode to build, deploy, run, and archive the release version of your iOS app. Use the same method that you would use for any Xcode project. For more information, see Prepare for app distribution in the XCode Help.

To generate a build

1. On your macOS computer, open Xcode.
2. In Xcode, navigate to your project and then choose Edit Scheme.
3. On the project page, for Build Configuration, select release.
4. Build your project as you would any Xcode project.
5. In the menu, choose **Product, Archive**.
6. In the dialog box, on the **Archives** page, click **Export**.
7. Generate your iOS app.

**Using Virtual File System with iOS**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

If you frequently change assets, you can configure Asset Processor on your macOS computer to use the virtual file system (VFS) to serve asset files to your iOS app. When you modify an asset, it is automatically reloaded without restarting the app. This method may result in slower load times because assets are sourced directly from your computer instead of the mobile device.

**Prerequisites**

Before you can use the VFS with iOS, you must download the usbmuxconnect package and save to a location on your macOS computer.

**Setting up VFS**

On your macOS computer, set up VFS to enable asset processing for iOS.

**To set up VFS on a macOS computer**

1. Do the following to configure Asset Processor for iOS:
   a. Navigate to the `lumberyard_version\dev\` directory.
   b. Use your preferred text editor to open the `AssetProcessorPlatformConfig.ini` file.
   c. Remove the preceding semicolon to uncomment `ios=enabled`.

   ```ini
   [Platforms]
   pc=enabled
   ;es3=enabled
   ios=enabled
   
   d. Save the file.

2. Do the following to update the bootstrap configuration settings:
   a. Navigate to the `lumberyard_version\dev\` directory.
   b. Use your preferred text editor to open the `bootstrap.cfg` file.
   c. Do the following:
      i. Set `remote_filesystem` to `1`.
      ii. Set `ios_connect_to_remote` to `0`.
      iii. Set `ios_wait_for_connect` to `1`.
      iv. Set `white_list` to the IP address of your macOS computer.

   **Note**
   Be sure to update the existing entries. Do not duplicate the entries.

3. Do the following to modify the system configuration file:
   a. Navigate to the `lumberyard_version\dev\` directory.
b. Use your preferred text editor to open the system_ios_ios.cfg file.
c. Set the r_AssetProcessorShaderCompiler console variable to 1.
d. Save the file.

4. Do the following to connect Asset Processor to the iOS device through the local host:
   a. Navigate to the lumberyard_version\dev\BinMac64 directory and launch Asset Processor.
   b. In Asset Processor, on the Connections tab, click Add Connection.
   c. For the new connection, select the Auto Connect check box. You can leave the default values for IP and Port.

5. In a Terminal window, navigate to the location where you saved the usbmuxconnect package.
6. Type the following: ./itnl --iport 22229 --lport 22229
7. Verify that Asset Processor displays one entry on the Connections tab with a status of Connected.

   Note
   If you receive a bind error message, please restart your macOS computer and try again.

Running the VFS Build

Run the VFS build on your macOS computer.

To run the VFS build

1. Do the following to update the bootstrap configuration settings:
   a. Navigate to the lumberyard_version\dev\ directory.
   b. Use your preferred text editor to open the bootstrap.cfg file.
   c. Do the following:
      i. Set remote_filesystem to 1.
      ii. Set ios_connect_to_remote to 0.
      iii. Set ios_wait_for_connect to 1.

2. Build and launch your game for iOS with Xcode. For instructions, see Creating a Release App (p. 3204).
3. Verify that Asset Processor displays an iOS connection on the Connections tab, and that your game runs on the device.

   Note
   If the device cannot be reached or you don’t see a connection in Asset Processor, stop the game, disconnect and then reconnect the device, and start again from step 1. If remote shader compilation is interrupted, you may see malformed shaders cached to your Windows computer. The malformed shaders can cause issues with subsequent runs using VFS. To resolve this issue, delete the lumberyard_version\dev\Cache\game_project_name \ios\user\ directory on your Windows computer. Then, restart the shader compiler.

Universal Remote Console

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Console commands to modify and configure the Lumberyard runtime application. On a PC, the Console is available from Lumberyard Editor or the game. But for mobile platforms you must use a separate Windows-based application called the Universal Remote Console. With the Universal
Remote Console you can use the IP address of the machine running the Lumberyard game to connect to a remote instance of Lumberyard.

Universal Remote Console requires the use of a PC and works with both Android and iOS. Your mobile device and the PC will need to be on the same network and your firewall should be configured to allow traffic through port 4600.

To start the Universal Remote Console

1. Run `lumberyard_version\dev\Tools\RemoteConsole\RemoteConsole.exe`

2. To see output from the Lumberyard logging system, click the **Full Log** tab.

To connect to a Lumberyard game on a mobile device

1. In the `system_platform_asset.cfg` file, enter the console variable
   `log_RemoteConsoleAllowedAddresses=[IP address]`, where IP address is the address of the computer that the remote console is running on.

   For Android, use the `system_android_es.cfg` file.

   For iOS, use the `system_ios_ios.cfg` file.

2. Save the file.

3. After you update the `.cfg` file, deploy the game to mobile devices.

   For more information, see [Deploy with Lumberyard Editor (p. 3177)](https://example.com) and [Creating a Release App (p. 3204)](https://example.com).

4. If the game is already running, restart it.

5. In the Universal Remote Console, click **Targets** on the toolbar.

6. Enter the IP address of the device under **Custom IP**.
If your network allows you to assign fixed IP addresses per device, you can edit the `params.xml` file and add the new target devices, as illustrated in the following example. This file is located in the same directory as Universal Remote Console, and you can edit it with the application running.

```xml
<Targets>
  <Target name="PC" ip="localhost" port="4600"/>
  <Target name="Android" ip="192.168.1.247" port="4600"/>
</Targets>
```

This lets you select from a list of devices instead of entering the IP address each time. Once successfully connected, the status indicator in the lower right corner will turn green.

### Issuing Commands

In the **Type a command** box at the bottom of the window, enter a command like the ones that follow. This control features autocomplete and, for certain commands (like `map`), can also detect available options.

Commands include the following:

- `cl_DisableHUDText` – Disables HUD text
- `g_debug_stats` – Enables gameplay events debugging
- `r_DisplayInfo` – Displays rendering information
- `r_ProfileShaders` – Displays profiling information for the shaders

### Sharing Assets Between Windows and macOS

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. [Download O3DE](#) or visit the AWS Game Tech blog to learn more.

You can build your assets on a Windows computer and then use them in an iOS build. To do so, you must set up sharing between your Windows and macOS computers.

#### To set up your Windows computer for sharing assets

1. On your Windows computer, navigate to the `lumberyard_version\dev\Cache\` directory.
2. Right-click the `Cache` directory and choose **Properties**.
3. In the **Properties** window, on the **Sharing** tab, click **Advanced Sharing**.
4. In the **Advanced Sharing** window, select the **Share this folder** check box.
5. (Optional) Click **Permissions** to set permissions for specific groups or users. Add users who will modify the files on a macOS computer.
6. Click **OK**.

#### To set up your macOS computer for sharing assets

1. On your macOS computer, rename any `Cache` directories. For example, rename to `Cache.old`.
2. In the Finder, choose **Go**, **Connect to Server**.
3. For **Server Address**, type: `smb://<IP address or DNS name of Windows computer>/Cache`
4. Click **Connect**.
5. (Optional) To enable your macOS computer to connect to this shared folder on startup, do the following:
a. Launch **System Preferences** and then choose **Users & Groups, Login Items**.

b. Click the + button to add a login item.

c. In the Finder, under **Shared**, select your Windows computer.

d. Select your shared **Cache** directory and then click **Add**.

e. In a Terminal window, navigate to the **lumberyard_version\dev\** directory.

f. To create a symbolic link to the shared **Cache** directory, type: `sudo ln -s /Volumes/Cache`

g. If prompted, type your macOS password.

---

**Add iOS Frameworks to a Lumberyard project**

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

As part of making changes to your core application on iOS, you may need to include bundled libraries for Darwin platforms called **Frameworks**. Frameworks are versioned bundles of libraries and their headers packaged as a single redistributable, and more common to ship for Darwin-based platforms rather than independent `.dylib` libraries and header files. When building for iOS, these frameworks need to be included in your final application bundle and code-signed like the rest of your binary. Because headers ship as part of a Framework, you don't need to explicitly export them so that the build system or your code will see the headers.

**Include frameworks in the launcher's wscript**

This section of the guide walks you through configuring your project launcher's wscript to support linking and including Frameworks, as well as . Before reading the rest of this topic, make sure that you're familiar with the general principles behind Waf build scripts in Lumberyard. For more information, see Waf Module Files (wscript) (p. 76).

1. Open the file *lumberyard_install_dir/dev/Code/LauncherUnified/wscript* in a text editor.

2. Inside the function `package(pkg)`, add the `custom_frameworks` argument when calling `pkg.package_game()`. The value of this argument is an array of paths to the frameworks that you want to include. Paths to frameworks can either be absolute or relative to *lumberyard_install_dir/dev*.  

   ```python
   pkg.package_game(
       target = '{}Launcher'.format(pkg.project),
       task_gen_name = '{}ClientLauncher'.format(pkg.project),
       platforms = ['appletv', 'darwin', 'ios'],
       client_only = True,
       exclude_test = True,
       custom_frameworks = [ path-to-framework-1, path-to-framework-2, ... ],
   )
   ```

3. Save and close the file.

4. Unless you need to also modify your `rpath` as described in the next section due to non-default framework locations, build your project.

During the build process, frameworks are copied to the @executable_path/Frameworks directory within your application bundle.
Add library search paths

If you've placed Frameworks in a location other than @executable_path/Frameworks, you need to add this directory to your project's rpath. iOS uses rpath information to find dynamic libraries to load at runtime, meaning that if your framework isn't included as a subdirectory of an rpath element, your application won't be able to load the framework library and may crash or experience other serious errors.

To add an element to your project's rpath, follow these instructions.

1. Open the file lumberyard_install_dir/dev/Code/LauncherUnified/wscript in a text editor.
2. Inside the function build(bld), add values to the ios_rpath argument when calling bld.CryLauncher(). The value of this argument is an array of rpath values to be used at runtime. rpath values should always start with @executable_path, which resolves to the root of the application bundle.

```python
bld.CryLauncher(
    # Common
target = 'ClientLauncher',
file_list = 'launcher.waf_files',
pch = 'Launcher_precompiled.cpp',
client_only = True,
exclude_test = False,
use = ['AzGameFramework'],
includes = [bld.Path('Code/CryEngine/CryCommon')],

    # Platform Specific
platform_roots = [{'root': 'Platform', 'export_includes': False}],
ios_rpath = [@executable_path/Frameworks, my-framework-directory, ...],
test_file_list = ['launcher_test.waf_files'],
test_use = ['AzTest'],
test_uselib = ['GMOCK'],
test_win_linkflags = ['/SUBSYSTEM:CONSOLE']
)
```

Important

Don't remove any values from this argument that you didn't add yourself! Removing any of the default rpath values will cause your application to crash.

3. Save and close the file.
4. Rebuild your project.

iOS Debugging and Troubleshooting

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard provides full access to the source code, which allows you to debug your iOS application using Xcode without additional Lumberyard-specific steps to follow. For information about debugging and profiling your iOS application, see Debugging in the official Apple developer documentation.

Unable to see activity in the shader compiler window

You must connect to the shader compiler on your PC in order to compile the subset of shaders required by your game, on demand. To verify that your app has connected correctly and obtained all shaders, you can view the output in the shader compiler window. If you still do not see any activity in the window,
please check your setup by following the instructions on the Run the Remote Shader Compiler (p. 3203) page.

**Assets appear out of date on iOS devices**

When you make and save changes to your project in Lumberyard Editor, these changes are automatically reflected on your iOS device the next time you deploy. Ensure you have set up your cache folder to share between your PC and Mac. If you encounter Xcode errors when deploying to your iOS devices or your assets appear out of date on the iOS devices, you can try cleaning your product from Xcode (click **Product, Clean**), which clears the .app package built to BinIos or BinIos.Debug (debug builds) in the directory where you installed Lumberyard.

Cleaning the project does not create a full rebuild of the iOS application

Lumberyard uses a custom build step to generate the final executable and temporary C++ object files, which output to the \BinTemp\ios_debug or \BinTemp\ios_profile directory where you installed Lumberyard. Unlike a regular Xcode project, in order to create a full rebuild of the iOS application, you must manually delete the contents of the output folder or run one of the following Waf commands from a Terminal window:

- To build debug, run the following command: `lmbr_waf.sh clean_ios_debug`
- To build profile, run the following command: `lmbr_waf.sh clean_ios_profile`
- To build release, run the following command: `lmbr_waf.sh clean_ios_release`

**Observed frame rate varies greatly**

While running your iOS application, the observable frame rate can vary depending on the build (debug or profile) you are running, whether you are connected to the Xcode debugger, and whether Metal API validation is enabled. To display the frame rate in the upper right corner of the screen, set the `r_DisplayInfo` configuration variable to 1 or higher. When your Xcode project is generated, the default build scheme is set up for debugging. If you want to test or profile your application's speed, we recommend that you edit your active scheme to run a profile build. Deselect **Debug executable** and disable **Metal API Validation**. Additionally, set the target resolution using the `r_WidthAndHeightAsFractionOfScreenSize` console variable or the `r_width` and `r_height` console variables in the system_ios_ios.cfg file. The default value is 1; however, you can lower the target render resolution to help improve performance. If the target render resolution is lower than the default (native device resolution), Lumberyard uses an anti-aliasing algorithm to help maintain the same level of visual quality as the native resolution.

**Design Considerations for Creating Mobile Games Using Lumberyard**

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard is a cross-platform game engine, which allows you to develop your game with less concern about the release platform(s). However, some mobile development considerations are discussed below, including game logic, input, and application lifecycle.
Application Lifecycle

Lumberyard provides a Process Life Management Gem (in the Project Configurator) that shows how you can respond to various application lifecycle events in order to pause your game, display a modal splash screen, and any other actions that need to occur if your application loses focus. You can access system-specific events in C++ by connecting to the appropriate EBus; however, Lumberyard also generates platform-agnostic events that you can use for all supported platforms.

<table>
<thead>
<tr>
<th>Lumberyard Application Lifecycle Events</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnApplicationConstrained</td>
<td>applicationWillResignActive</td>
<td>onPause()</td>
</tr>
<tr>
<td>OnApplicationUnconstrained</td>
<td>applicationDidBecomeActive</td>
<td>onResume()</td>
</tr>
<tr>
<td>OnApplicationSuspended</td>
<td>applicationDidEnterBackground</td>
<td>onPause()</td>
</tr>
<tr>
<td>OnApplicationResumed</td>
<td>applicationWillEnterForeground</td>
<td>onResume()</td>
</tr>
<tr>
<td>OnMobileApplicationWillTerminate</td>
<td>applicationWillTerminate</td>
<td>onDestroy()</td>
</tr>
<tr>
<td>OnMobileApplicationLowMemoryWarning</td>
<td>applicationDidReceiveMemoryWarning</td>
<td>onLowMemory()</td>
</tr>
</tbody>
</table>

To receive process lifecycle events in your game

1. Derive your class from AzFramework::ApplicationLifecycleEvents::Bus::Handler (or AzFramework::[Ios|Android|Windows]LifecycleEvents::Bus::Handler for platform-specific events).
2. Override the functions corresponding to the events you wish to override:

```cpp
void OnApplicationConstrained(Event /*lastEvent*/) override;
void OnApplicationUnconstrained(Event /*lastEvent*/) override;
void OnApplicationSuspended(Event /*lastEvent*/) override;
void OnApplicationResumed(Event /*lastEvent*/) override;
```

3. Connect to the event bus when you want to start listening for events (be sure to also disconnect when you no longer wish to receive them):

```cpp
ApplicationLifecycleEvents::Bus::Handler::BusConnect();
???
ApplicationLifecycleEvents::Bus::Handler::BusDisconnect();
```

For a complete example of how to subscribe and respond to application events, see the Gems\ProcessLifeManagement\Code\Source\ProcessLifeManagementGem.h\.cpp directory.

Lumberyard Performance Tuning Guidelines for Mobile Devices

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
See the following guidelines for generating art assets for mobile games with Lumberyard and how to tune Lumberyard performance for mobile devices.

**Topics**
- Art Guidelines (p. 3216)
- Engineering Guidelines (p. 3220)
- GPU Memory (GMEM) (p. 3221)
- Memory and App Size (p. 3223)
- Particle Rendering Features and Performance (p. 3224)

**Art Guidelines**

This section provides guidelines for geometry, lighting, cascade shadow maps, and materials.

**Geometry Guidelines**

Follow these guidelines when creating geometry for your mobile game:

- Use fewer individual objects to significantly reduce the draw calls. We recommend 750 or less draw calls on mobile devices.
- Use instances where appropriate. To do so, create a slice in Lumberyard Editor and then instantiate multiple instances of that slice in the level.
- Add support for geometry level of detail (LOD). Each LOD should be a 50% reduction of geometry.
  - Group the high resolution LOD model under \_lod0\_objectname.
  - Group the next LOD model under \_lod1\_objectname\_group.
  - Name the shadow proxy model as shadowproxy.
  - Export the shadow proxy model with the shadow proxy material.
  - Group the shadow proxy model under the last LOD group.

**Example**

The following is an LOD setup in Maya.
• Do the following to tune the LOD:
  1. In Lumberyard Editor, select an entity in your level that has a Mesh (p. 684) component.
  2. In the Entity Inspector, in the Mesh component, under Options, adjust the slider for LOD distance ratio. The slider adjusts the distance at which the LOD pops in.
  3. Tune each asset to see only the high resolution asset when needed.

• Restrict your poly count as follows for each camera view in the game:
  • For low-end devices, use 300k or below.
  • For higher-end devices, use 700k or below if the CPU is the bottleneck.

  **Note**
  To determine the poly count, set r_DisplayInfo to 1 in the Lumberyard Editor console. The debug data on the top right of the screen will display the poly number for each frame of the scene.

• In the Console pane in Lumberyard Editor, set the console variable r_stats to 1 to print the number of draw calls and polygons that the current camera renders.

**Lighting Guidelines**

Follow these guidelines when adding lighting to your mobile game:

• Carefully consider the amount of lights in your scene that cast shadows.

• Do the following to tune the lights:
  1. In Lumberyard Editor, select an entity in your level that has a Mesh component.
  2. In the Entity Inspector, in the Mesh component, under Options, adjust the value for View distance multiplier. Specify a lower value to fade out the light and stop drawing sooner. This setting is important for performance.

• Add area and projector lights only if needed. Lighting pass is expensive.

• Reduce the light area if possible. Be frugal.

• Avoid overlapping light areas.

• Avoid overlapping environment probe areas.

• Avoid covering large areas with environment probes, which can introduce artifacts on the edge of the large areas. For example, 1,000 x 1,000 is too large for an environment probe to cover. We recommend dividing an area that size into four smaller areas.

• Simulate bounced lighting with lights that don't cast shadows. Global illumination cannot be used on mobile devices.

  **Example**

  The following scene uses non-shadow casting lights to simulate global illumination (GI) bounce.
Cascade Shadow Map Guidelines

Lumberyard uses console variables to specify how to generate cascade shadow maps and to improve performance for the shadow pass. You can also set these variables to impact the engine globally or per level. Edit the level.cfg file to set the variables for a specific level. For more information, see Using the Console Window (p. 210).

- e_ShadowsCascadesDebug – Enables the debug view for the cascade shadow maps. Each cascade in the world renders with a different color to provide visual feedback of the area that is covered by a cascade shadow map.

Example

The following global shadow map (GSM) has the e_ShadowsCascadesDebug console variable set to 1.
Use the following console variables to specify the size and how to generate shadow cascades:

- **e_GsmLodsNum** – Specifies the number of shadow cascades to use. The default value is 5.
- **e_GsmRange** – Specifies the size of the first shadow cascade in square world units. The default value is 3, which covers three square units in the world. In the previous example, the first shadow cascade is drawn in red.
- **e_GsmRangeStep** – Specifies the multiplier to use to calculate the size of the next shadow cascade. The default value is 3.

For example, if you set `e_GsmRange` to 3, the first shadow cascade covers three square units in the world. To calculate the next shadow cascade size, Lumberyard multiplies the first shadow cascade and the value for `e_GsmRangeStep`. If the value for `e_GsmRangeStep` is the default value of 3, the next shadow cascade would cover nine square units in the world. The next shadow cascade would cover 27 square units in the world, and so on.

When you set console variables, try to balance reducing the size and number of cascade shadow maps and maintaining the visual quality of your game. You can set `e_shadows` to 0 to turn off shadows, which may be necessary for low-end devices.

The following images demonstrate how the global shadow map (GSM) console variables affect the shadow cascade size and coverage distance.

---

**Example images for GSM impact on shadow cascade size and coverage distance**

<table>
<thead>
<tr>
<th>Description</th>
<th>Console Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the <code>e_GsmRange</code> console variable to 1 to reduce the size of the shadow cascade.</td>
<td><code>e_GsmRange</code></td>
<td>1</td>
</tr>
<tr>
<td>Set the <code>e_GsmRangeStep</code> console variable to 1.5 to reduce the area that each consecutive cascade covers.</td>
<td><code>e_GsmRangeStep</code></td>
<td>1.5</td>
</tr>
<tr>
<td>Set the <code>e_GsmLodsNum</code> console variable to 3 to reduce the number of shadow cascades that Lumberyard uses.</td>
<td><code>e_GsmLodsNum</code></td>
<td>3</td>
</tr>
</tbody>
</table>
Material Guidelines

Follow these guidelines when creating materials for your mobile game:

- Use fewer individual objects and materials to significantly reduce the draw calls.
- Use texture atlases to reduce the number of materials or submaterials needed, reduce the draw calls, and increase performance.
- Reduce the texture size to 1024 x 1024 or less.

Engineering Guidelines

Lumberyard provides four levels of configuration files to support enabling and disabling features and functionality based on performance characteristics of the mobile devices. You can find the following files in the `lumberyard_version/dev/Engine/Config/spec` directory:

- `ios_low.cfg`
- `ios_medium.cfg`
- `ios_high.cfg`
- `ios_veryhigh.cfg`
- `android_low.cfg`
- `android_medium.cfg`
- `android_high.cfg`
- `android_veryhigh.cfg`

Each file includes a set of console variables in which you can enable or disable engine features.

You can also edit the configuration files in the Graphics Settings window in Lumberyard Editor.

To edit configuration files in Lumberyard Editor

1. In Lumberyard Editor, choose Edit, Editor Settings, Graphics Settings.
2. In the Graphics Settings window, do the following:
   a. For Platform, select your device such as iOS.
   b. For the preferred configuration file, adjust the values for the various console variables. For example, if you select iOS, console variables and values for each configuration file appear (ios_low.cfg, ios_medium.cfg, ios_high.cfg, and ios_veryhigh.cfg).

When you edit the console variables for the configuration file, the renderer displays onscreen how the level may look on a mobile device.
Using android_models.xml and ios_models.xml Files

Lumberyard uses two .xml files to determine which mobile devices use the low, medium, high, or very high configuration settings files. You can find the android_models.xml and ios_models.xml files in the /lumberyard_version/dev/Engine/Config/gpu directory.

Example android_models.xml file

The Samsung Galaxy S5 line shows that Lumberyard supports the use of regular expressions when you specify the device model number. If a device model is listed more than once, the last instance is read and used with the corresponding configuration file.

```xml
<DeviceList>
  <Config file="android_low.cfg">
    <Device model="KFTHWI"/> <!-- Amazon Kindle Fire HDX -->
    <Device model="Nexus 7"/> <!-- Nexus 7 -->
    <Device model="Nexus 5"/> <!-- Nexus 5 -->
    <Device model="SM-N910H"/> <!-- Samsung Galaxy Note 4 (Asia-Pacific)-->
    <Device model="SM-G900\w{1,2}"/> <!-- Samsung Galaxy S5 -->
  </Config>
  <Config file="android_medium.cfg">
    <Device model="SM-T217S"/> <!-- Samsung Galaxy Tab 3 -->
    <Device model="SM-N920C"/> <!-- Samsung Note 5 -->
  </Config>
  <Config file="android_high.cfg">
    <Device model="Nexus 6"/> <!-- Motorola Nexus 6 -->
    <Device model="Pixel"/> <!-- Google Pixel -->
  </Config>
  <Config file="android_veryhigh.cfg">
  </Config>
</DeviceList>
```

GPU Memory (GMEM)

GPU memory (GMEM) is a class of optimizations that uses local memory on the GPU to reduce the transfer of large textures between the CPU and GPU. GMEM can operate in the following modes, depending on what the mobile device can support:
• 256 bit mode – The engine can store three GBuffer render targets and depth or stencil in the local GPU pixel memory, while doing the preliminary Z pass, generating the GBuffer, deferring decals, and deferring rain and snow passes. This mode also stores the diffuse and specular light accumulation textures in the local GPU pixel memory during the deferred shading passes of the renderer.

• 128 bit mode – This mode stores the diffuse and specular light accumulation textures in the local GPU pixel memory during the deferred shading passes of the renderer. Compared to the 256 bit mode, the 128 bit mode provides less of a performance boost. However, with the 128 bit mode you can enable certain rendering features that are not available if you use the 256 bit mode.

• Disabled – This mode means that GMEM is not supported or has been disabled in the configuration files.

The renderer uses two OpenGL extensions to determine which GMEM mode is supported for Android devices:

• Framebuffer fetch (256 bit mode)
• Pixel Local Storage (128 bit mode)

For the iOS devices that Lumberyard supports, both GMEM 256 bit mode and 128 bit mode are supported.

Setting the GMEM Mode

You can enable or disable GMEM with the r_EnableGMEMPath console variable.

• 0 = Disables GMEM in the renderer.
• 1 = Enables the renderer to use GMEM in 256 bit mode. If the mobile device doesn't support 256 bit mode, the renderer will use 128 bit mode. If the mobile device doesn't support 128 bit mode, the renderer will disable GMEM.
• 2 = Enables the renderer to use GMEM in 128 bit mode.

You can only enable or disable GMEM during engine startup. You must add the r_EnableGMEMPath console variable to one of the following files:

• android_low.cfg
• android_medium.cfg
• android_high.cfg
• android_veryhigh.cfg
• ios_low.cfg
• ios_medium.cfg
• ios_high.cfg
• ios_veryhigh.cfg
• system_android_es3.cfg
• system_ios_ios.cfg

Note
To prevent visual artifacts and performance issues, do not change the value of the r_EnableGMEMPath console variable during runtime.

Rendering Features and GMEM

GMEM offers the flexibility of setting the mode and rendering features to meet the visual quality and performance for a range of mobile devices. More powerful mobile devices may be able to use GMEM
128 bit mode and still meet the necessary performance requirements. You can use the configuration files for your mobile device as well as the `android_models.xml` or `ios_models.xml` files to set your requirements.

If you enable GMEM 256 bit mode, you cannot use the following features:

- Motion blur
- Temporal antialiasing
- Screen space reflections
- Screen space directional occlusion
- Fur
- Deferred subsurface scattering
- Volumetric fog
- Deferred rain or snow occlusion

If you enable GMEM 128 bit mode, you cannot use the following features:

- Fur
- Deferred subsurface scattering
- Volumetric fog
- Deferred rain or snow occlusion

If you want to use a rendering feature that GMEM 128 bit mode supports but not GMEM 256 bit mode, the renderer automatically sets the mode to 128. This is true even if the mode is set to 256 in the configuration files.

To use all available rendering features, disable GMEM.

### Memory and App Size

Because mobile devices have a limited amount of memory, you should take the necessary steps to reduce the size of your app and the amount of memory it requires. Follow these guidelines:

- Include only the resources that your game uses.
  - Enable only the gems that are required for your game. This will prevent unnecessary code and assets from being bundled into your app.
- Include only the assets that your game uses.
  - Check the `Cache` directory for your game project to see which assets the build process will pack into the app bundle.
  - On Windows, use WinDirStat to identify which assets use the most resources, and determine whether to include the assets in the app bundle or host the assets online as a separate download.
  - Remove editor-only assets, which Asset Processor adds to the `Cache` directory. This will reduce the size of the final app bundle.
  - Remove assets in the engine directory. These assets may significantly increase the size of your final app bundle, even if your game doesn't use them. The `defaulttextures.xml` file contains a list of textures that certain game projects might not use. You can find this file in the `/lumberyard_version/dev/Engine/EngineAssets` directory.
- Avoid using static allocations. This will reduce the size of your executable code and allow for safer memory management at runtime.
- Provide downloads for region-specific assets. If your app uses localized content and you plan to deploy to multiple regions, consider dividing up your app and hosting the data for each region on Amazon S3. You can then use the dynamic content system to download region-specific assets.
Android

If you want to reduce the size of a large app, you can use the tools that are included with the Android NDK to examine your executable. For example, you can use objdump or nm. The nm tool can disassemble your binary files and show the size of each code segment. The nm tool can also list symbols and detect if code is unexpectedly linked to your binary.

Particle Rendering Features and Performance

Particles use sun and light volumes to determine how they should be lit in the scene. Because light volumes are expensive on mobile devices, we do not recommend using this feature on medium to low-end devices. To specify how particles are lit, you can use the e_LightVolumes console variable with the following values:

- 0 = Particles are not affected by the sun or light volume lights.
- 1 = Particles are affected by the sun and light volume lights.
- 2 = Particles are affected only by the sun light.

Modifying Project Settings for Mobile Device Games

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the Project Settings Tool (PST) to make streamlined changes to project settings across all mobile platforms. The Project Settings Tool shows properties from all of the project settings files so that you can edit and save changes at once.

Note

- Before you can use the Project Settings Tool, you must have Lumberyard Editor installed, and an active project created and set as the default project.
- Currently, only Android and iOS are supported for platform-specific configuration.

Topics

- Settings Files (p. 3224)
- Using the Project Settings Tool (p. 3225)
- Properties (p. 3229)

Settings Files

The Project Settings Tool modifies project settings files which are located in their respective project directories. The main settings file, project.json (p. 69), is located in the root directory of each project and contains cross-platform settings for platforms such as PC and Android. The file also contains platform-specific settings for PC and Android.

You can find the project settings for iOS in project_name\Root\Gem\Resources\PlatformLauncher\Info.plist. Plist files have a special format of XML that use dictionaries to store properties. All plist files have some properties that are common across platforms, yet are stored in each individual file.

For more information about .plist files, see About Info.plist Keys and Values and Core Foundation Keys.
For more information about iOS-specific settings, see iOS Keys.

Using the Project Settings Tool

You can use the Project Settings Tool to set settings related to the current project.

To open the PST

1. In Lumberyard Editor, choose File, Project Settings, Project Settings Tool.
2. In the Project Settings Tool, you can review and change your settings. For more information, pause on a property and review the tooltip. You can also see the section called “Properties” (p. 3229).
   • The Base Settings apply to all platforms.
   • The Platform Settings section has tabs for platform-specific settings.
3. If you haven’t saved your changes, and you want to reload the settings back to the way they are on disk (in your settings files), choose Reload.
Image Previews

For image settings such as Icons and Splashscreens, the Project Settings Tool displays image previews using the image to be used for each dpi or size.
Validation

The Project Settings Tool validates settings as you enter values and provides feedback when a value is invalid with the selected setting.

A red outline appears around the setting to indicate an invalid value. You can also pause on the setting to view the error message that describes the reason that the field is invalid.

Linked Properties

You can link similar properties to each other. When you link properties to one another, modifying one of them makes the same changes to all other linked properties.

To work with link properties

1. In the Project Settings Tool, navigate to the property that you want to link. Properties that you can link have a chain-link icon.
2. Pause on the link icon to see what properties are linked. Properties that you can link are automatically linked when loaded from the settings file if all relevant properties have the same value.
3. You can enable or disable any link. To enable a link, click the icon. All properties that it links to are updated with the current value.

   When a property's link is enabled and linked to a valid property, the link icon appears intact and has an orange outline.

   ![Image 1](image1.jpg)

   When a property's link is broken or disabled, the link icon appears broken and without an orange outline.

   ![Image 2](image2.jpg)

4. To disable a link, click the icon again.

   **Note**
   Some properties are always linked and can't be disabled, such as **Base Settings - Project Name** and **iOS - Bundle Name**. This ensures accuracy for properties that are required to have the same values across platforms. If the **Project Settings Tool** finds discrepancies between settings files for the always-linked properties, the **project.json** values take precedence.

**Reconfiguring the Project**

After you make changes and save, the **Project Settings Tool** prompts you to reconfigure your project.

![Image 3](image3.jpg)

If you choose to run the **configure** command, the output appears at the bottom of the window. Once the results display "Reconfiguration Finished," you can use the **Deployment Tool** to deploy to your device with the new changes.

**Note**
If you want to deploy your changes immediately, choose **Yes**. Lumberyard Editor doesn't automatically run the **configure** command and will not prompt you to do so later.
Properties

See the following properties in the Project Settings Tool.

Base Settings

The Base Settings properties apply to both Android and iOS.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Name</td>
<td>Name that Lumberyard uses to identify the selected project. This value should not be changed.</td>
</tr>
<tr>
<td>Product Name</td>
<td>Name to be displayed in titles for the executables or apps.</td>
</tr>
<tr>
<td>Executable Name</td>
<td>File name of the application that runs the project. This value should not be changed.</td>
</tr>
<tr>
<td>Game Folder</td>
<td>Directory name that stores all of the project's code and resources. This value should not be changed.</td>
</tr>
<tr>
<td>Game Dll Name</td>
<td>File name of the DLL used to load the game. This value should not be changed.</td>
</tr>
<tr>
<td>Output Folder</td>
<td>Directory to which the packaged project is exported after being built.</td>
</tr>
<tr>
<td>Code Folder</td>
<td>A legacy setting that specifies the directory that contains code for the project.</td>
</tr>
</tbody>
</table>

Android Settings

You can find the following properties in the Platforms Settings on the Android tab.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Name</td>
<td>Android application package identifier. Used for generating the project-specific Java activity class and in the AndroidManifest.xml. Must be in dot-separated format.</td>
</tr>
<tr>
<td>Version Name</td>
<td>Human-readable version number. Used to set the android: versionName tag in the AndroidManifest.xml and is displayed in Google Play.</td>
</tr>
<tr>
<td>Version Number</td>
<td>Internal application version number. Used to set the android:versionCode tag in the AndroidManifest.xml. This value must always be greater than the previously submitted build. Otherwise, submission to Google Play will fail.</td>
</tr>
<tr>
<td>Orientation</td>
<td>Desired orientation of the Android application. Used to set the android:screenOrientation tag in the AndroidManifest.xml.</td>
</tr>
<tr>
<td>Public App Key</td>
<td>Application license key provided by Google Play. Required for using APK expansion files or other Google Play Services.</td>
</tr>
<tr>
<td>App Obfuscation Salt</td>
<td>Application-specific salt value for (un)obfuscation when using APK expansion files.</td>
</tr>
<tr>
<td>Rc Job PAK Override</td>
<td>Path to the RC job XML file used to override the normal PAK files generation used in release builds. Path must be relative to dev\Bin64\rc.</td>
</tr>
<tr>
<td>Rc Job APK Override</td>
<td>Path to the RC job XML file used to override the normal APK Expansion file(s) generation used in release builds. Path must be relative to dev\Bin64\rc.</td>
</tr>
<tr>
<td>Use Main APK</td>
<td>Specifies whether the Main APK Expansion file should be used.</td>
</tr>
<tr>
<td>Use Path PAK</td>
<td>Specifies whether the Patch APK Expansion file should be used.</td>
</tr>
<tr>
<td>Enable Screen Wake Lock</td>
<td>Enables the screen wake lock. If enabled, the device won’t go to sleep while the app is running.</td>
</tr>
<tr>
<td>Disable Immersive Mode</td>
<td>Disables hiding of the top and bottom system bars.</td>
</tr>
<tr>
<td>Icons</td>
<td>All icon overrides for Android. The Default property is used on all overrides that are not explicitly specified. PNG images are required.</td>
</tr>
</tbody>
</table>
### Property | Description
--- | ---
Resolutions | Resolutions must be 48px, 72px, 96px, 144px, and 192px.

**Splashscreens**

All splashscreen (the image shown while the app first loads) overrides for Android. The **Default** property is used on all overrides that are not explicitly specified. PNG images are required.

Resolution values are not strictly enforced but the recommended values are 1024 x 640, 1280 x 800, 1920 x 1200, and 2560 x 1600.

### iOS Settings

You can find the following properties in the **Platforms Settings** on the **iOS** tab.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bundle Name</strong></td>
<td>Internal name of the bundle that iOS uses to identify it. This value should not be changed.</td>
</tr>
<tr>
<td><strong>Display Name</strong></td>
<td>User-visible name of the app that is shown in the app store.</td>
</tr>
<tr>
<td><strong>Executable Name</strong></td>
<td>Name of the bundle's generated XCode project. This value should not be changed.</td>
</tr>
<tr>
<td><strong>Bundle Identifier</strong></td>
<td>Uniquely identifies the bundle on iOS. This value should be in reverse-DNS format.</td>
</tr>
<tr>
<td><strong>Version Name</strong></td>
<td>Release version string for the app that is displayed in the App Store.</td>
</tr>
<tr>
<td><strong>Version Number</strong></td>
<td>Build version number string for the bundle. This value must be greater than the previous submission to the App Store.</td>
</tr>
<tr>
<td><strong>Development Region</strong></td>
<td>Default language and region for the app. Defined by the region in which it was developed and the primary language used during development.</td>
</tr>
<tr>
<td><strong>Requires Fullscreen</strong></td>
<td>Specifies whether the app is required to run in full-screen mode.</td>
</tr>
<tr>
<td><strong>Hide Status Bar</strong></td>
<td>Specifies whether the status bar is initially hidden when the app launches.</td>
</tr>
<tr>
<td><strong>Orientations</strong></td>
<td>Select which orientations to enable on the iOS device:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Landscape (right home button)</strong> – Enable landscape orientation with home button on right side of device.</td>
</tr>
</tbody>
</table>
### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Landscape (left home button)</strong></td>
<td>Enable landscape orientation with home button on left side of device.</td>
</tr>
<tr>
<td>• <strong>Portrait (bottom home button)</strong></td>
<td>Enable portrait orientation with home button on bottom of device.</td>
</tr>
<tr>
<td>• <strong>Portrait (top home button)</strong></td>
<td>Enable portrait orientation with home button on top of device.</td>
</tr>
<tr>
<td><strong>Icons</strong></td>
<td>All icon overrides for iOS. PNG images are required. All resolutions must be exactly as specified.</td>
</tr>
<tr>
<td><strong>Launchscreens</strong></td>
<td>All launch screen overrides for iOS. PNG images are required. All resolutions must be exactly as specified.</td>
</tr>
</tbody>
</table>

### Override Images

On iOS devices, override images are stored in the `project_root\Gem\Resources \platform\Launcher\Images.xcassets`. Within that directory, the `AppIcon.appiconset` directory contains icons and the `LaunchImage.launchimage` directory contains splashscreens.

### Example

The following image shows the default iOS icons and splashscreens. When you choose an override for one of these images, it overwrites the currently selected one. The overwrite is completed when you choose **Save**. This change can't be undone.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard supports low, medium, high, and very high graphics performance for mobile devices. You can change the graphics settings in Lumberyard Editor to simulate the graphics on Android and iOS devices. This feature helps you visually check your mobile games in Lumberyard Editor during the development process.

Updating Graphics Performance in Lumberyard Editor

Use the Graphics Performance menu in Lumberyard Editor to choose a graphics setting for Android or iOS.

To update graphic settings in Lumberyard Editor

1. In Lumberyard Editor, choose Edit, Editor Settings, Graphics Performance.
2. Choose Android or iOS and then choose a setting. For more information, see Editor Settings (p. 189).
When you choose a graphics performance setting, the `r_GraphicsQuality` console variable automatically updates with a corresponding value:

- Very High: 4
- High: 3
- Medium: 2
- Low: 1

You can change the default value for the `r_GraphicsQuality` console variable in each OS configuration file. You can find the files in the `lumberyard_version\dev\Cache\game_project\pc` directory.

**Example iOS Configuration File**

The following is an example of the `system_ios_ios.cfg` file.
If you set \texttt{r\_GraphicsQuality} to 0, the graphics performance setting is automatically detected based on the mobile device model or total RAM. You can review the mapping for mobile device models and graphics performance settings in the \texttt{android\_models.xml} and \texttt{ios\_models.xml} files, located in the \texttt{lumberyard\_version\dev\Cache\game\_project\pc\game\_project\config} directory.

Example

The following is an example of the \texttt{ios\_models.xml} file.
Modifying Graphics Configuration Files

Use the Graphics Settings window in Lumberyard Editor to modify the graphics settings for each OS configuration file. You can also modify the configuration files for the following graphics settings: low, medium, high, and very high.

To modify a graphics configuration file

1. In Lumberyard Editor, choose Edit, Editor Settings, Graphics Settings.
2. In the Graphics Settings window, for Platform, choose Android or iOS for the configuration file that you want to modify.
3. Modify the graphics settings for each parameter as needed.

4. Click **Save**.

**Note**

The configuration file saves to the `lumberyard_version\dev\game_project\Config` directory. Asset Processor prioritizes the settings in this configuration file above the engine's default configuration file, located in the `lumberyard_version\dev\Engine\Config` directory.
Adding IP Addresses to Allow Access to the Asset Processor and Remote Console

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Asset Processor is a networked application that Lumberyard uses to build source assets into game engine ready assets. To ensure your external device can connect to the Asset Processor, you must add the IP address of the external device (Android or iOS) to the `white_list` in the `bootstrap.cfg` file (located in the `lumberyard_version\dev` directory).

The Universal Remote Console is a networked application that Lumberyard uses to send commands and view output from the running game engine. To ensure remote console access to a running game instance on your external device, you must add the IP address of the computer that will run the remote console to the `log_RemoteConsoleAllowedAddresses` list in the appropriate configuration file (located in the `lumberyard_version\dev` directory):

- Android – `system_android_es3.cfg`
- iOS – `system_ios_ios.cfg`

You must update the configuration file to include the allowed IP addresses before you deploy your game to the external device.

Running the Shader Compiler on Amazon EC2

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Amazon Elastic Compute Cloud (Amazon EC2) provides a GPU instance that you can use to run the Lumberyard shader compiler for mobile (Android and iOS) and macOS development. For example, you can use the AWS device farm to test a build rather than hosting a PC on a public IP address. Amazon EC2 also provides a G2 instance type that supports advanced rendering features such as texturing, shadows, and anti-aliasing. For more information, see Amazon EC2 Instances.

To run the Lumberyard shader compiler, you must do the following:

1. Set up the Amazon EC2 instance.
2. Install virtual network computing (VNC) software.
3. Connect to the shader compiler.

Topics

- Prerequisites (p. 3240)
- Setting Up the Amazon EC2 Instance (p. 3240)
- Installing VNC Software (p. 3240)
- Connecting to the Shader Compiler (p. 3241)
Prerequisites

To run the Lumberyard shader compiler on Amazon EC2, you must have the following:

- AWS account
- Familiarity with the AWS Management Console
- Understanding of Amazon EC2 instances, VNC, security groups, and the Lumberyard tool chain

Setting Up the Amazon EC2 Instance

Before you can set up the Amazon EC2 instance, you must request a GPU instance using the Request to Increase Amazon EC2 Instance Limit link. When you are done, follow the steps below. For information about Windows GPU instances, see Windows Accelerated Computing Instances.

To set up the Amazon EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the console dashboard, choose Launch Instance.
3. On the Amazon Machine Image (AMI) page, enter GPU.
4. In the search results, select Windows Server 2012 R2 with NVIDIA GRID GPU Driver.
5. On the Choose an Instance Type page, select the g2.2xlarge type.
6. Choose Review and Launch to let the wizard complete the other configuration settings for you.
7. On the Add Storage page, add a drive with sufficient space (512 GB minimum).
8. On the Add Tags page, add a memorable tag for the computer. For example, ShaderCompilerMachine.
9. On the Review Instance Launch page, create a new security group or use an existing one.
10. Open port number 61453 for the shader compiler.
12. When prompted for a key pair, select Choose an existing key pair, and then select the key pair that you created when getting set up.
13. When you are ready, select the acknowledgement check box, and then click Launch Instances.
14. A confirmation page lets you know that your instance is launching. Click View Instances to close the confirmation page and return to the console.

Installing VNC Software

Once the Amazon EC2 instance is set up, you can install virtual network computing (VNC) software to run the shader compiler on the instance.

To install VNC software

1. Using a remote desktop connection, log in to the Amazon EC2 instance.
   
   **Note**
   
   If you are using a Mac, you must download Remote Desktop App from the App Store.
2. Download and install your preferred VNC software.
3. On your Amazon EC2 instance, run the VNC server.
4. Terminate the remote desktop session.
5. On your local PC, install a VNC client.
6. Using the VNC connection, run the shader compiler on the Amazon EC2 instance.
7. Leave the VNC window open and then follow the steps in Connecting to the Shader Compiler (p. 3241).

Connecting to the Shader Compiler

Follow these steps to connect to the shader compiler.

To connect to the shader compiler

1. On your computer, open the platform configuration file:
   - For Android, open the `system_android_es3.cfg` file (located in the `lumberyard_version\dev` directory).
   - For iOS, open the `system_ios_ios.cfg` file (located in the `lumberyard_version\dev` directory).

2. Edit the configuration file to set `r_ShaderCompilerServer` to the public IP address of your Amazon EC2 instance. You can retrieve the IP address from the Amazon EC2 console.

3. Run your game on your device to connect to the EC2 instance and compile the shaders.

Using AWS Device Farm in Lumberyard Editor

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use AWS Device Farm to test your Android or iOS apps on many different types of devices without needing to have any of them present physically. Device Farm can help you find and resolve compatibility issues on a large number of devices. For more information about Device Farm, see the Device Farm Developer Guide.

Starting in Lumberyard version 1.20, you can use the Lumberyard Editor Deployment Tool (p. 3175) to make Device Farm a deployment target.
Prerequisites

To use Device Farm in Lumberyard Editor, you need the following requirements:

- The Cloud Gem Framework gem, enabled for your project in the Project Configurator.

For information about using Project Configurator to enable gems, see Enabling Gems (p. 1064).

- A set of Amazon Web Services (AWS) credentials and access keys.
- If you do not have an AWS account, complete the following steps to create one.
To sign up for an AWS account

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on
   the phone keypad.

   - For more information about access keys, see How do I create an AWS access key?.
   - For information about the AWS Free Tier, see the AWS Free Tier page.
   - Device Farm requires that you have a shader compiler running on an Amazon Elastic Compute
     Cloud (Amazon EC2) instance. For more information, see Running the Shader Compiler on Amazon
     EC2 (p. 3239).

Creating an AWS Profile in Lumberyard Editor

After you have a set of AWS credentials, use them to create a profile in Lumberyard Editor.

To enter your credentials in Lumberyard Editor

1. In Lumberyard Editor, choose AWS, Credentials manager.

2. In the Credentials Manager dialog box, click Add profile.

3. In the Add profile dialog box, enter the required information.
Using AWS Device Farm in Lumberyard Editor

1. For **Profile name**, enter a name for the profile.
2. For **AWS access key** and **AWS secret key**, enter the access key and secret key.

**Important**

Do not share these credentials with anyone, and do not check them into source control. These credentials grant control over your AWS account, and a malicious user could incur charges.

4. Click **Save**.
5. In **Credentials Manager**, click **OK**.

You have now created a profile that is associated with your credentials. It is saved locally on your machine in your AWS credentials file. This file is usually located in your `C:\Users\user_name\.aws` directory.

**Accessing the Device Farm Console from Lumberyard Editor**

After the new profile is selected in the **Credentials Manager**, you can use Lumberyard Editor to access the Device Farm console.

**To access the Device Farm console from Lumberyard Editor**

1. Choose **AWS, Credentials Manager** and ensure that a profile is selected.
2. Choose **AWS, Open AWS Console, Device Farm**. This signs you into AWS and creates a new session.

**Configuring a Deployment for AWS Device Farm**

After you have selected a valid AWS profile in Lumberyard Editor, you can use the **Deployment Tool** to configure a deployment to AWS Device Farm.

**To configure a deployment for AWS Device Farm.**

1. Ensure that a valid AWS profile is selected.
2. In Lumberyard Editor, choose **File, Project Settings, Deploy to device.**
3. In **Deployment Tool**, under **Deploy**, click the **AWS Device Farm** tab.
4. The following options are specific to Device Farm usage.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load current level</strong></td>
<td>Because the Deployment Tool cannot communicate directly with the actual devices used by Device Farm, this option is not supported. Instead, specify the level that you want to test in your project's <code>autoexec.cfg</code> file as a <code>map levelName</code> command.</td>
</tr>
<tr>
<td><strong>Shader compiler IP address</strong></td>
<td>Set the shader compiler IP address to the IP address of your EC2 instance shader compiler. For more information, see Running the Shader Compiler on Amazon EC2 (p. 3239).</td>
</tr>
<tr>
<td><strong>Project name</strong></td>
<td>Select a Device Farm project, or click Add to create a new project.</td>
</tr>
<tr>
<td><strong>Device pool name</strong></td>
<td>Select a device pool, or click Add to create a device pool. For more information about adding a device pool, see the next step.</td>
</tr>
<tr>
<td><strong>Execution timeout</strong></td>
<td>Specify the maximum amount of time, in minutes, that you want the test to run. Possible values are from 5 to 150.</td>
</tr>
<tr>
<td><strong>Test type</strong></td>
<td>Built-in fuzz, the only option, sets the event throttle to 1000. This means that the test waits one second between random input events. The event count is 600.</td>
</tr>
</tbody>
</table>

5. Adding a device pool opens the **Device Farm Device Pool** dialog box, which retrieves the latest list of supported devices from Device Farm.
6. For **Name**, enter a name for your device pool, and then select the devices that you want to test your application on.

7. Click **Save**.

**Building and Deploying an App to AWS Device Farm**

At this point, you are ready to build and deploy your app to Device Farm.
To build and deploy an app to AWS Device Farm

1. Click **Deploy to Device Farm**. The deployment advances through the following steps:
   a. Process all assets. This can take a long time if this is the first time assets for the target platform are processed.
   b. Run the WAF build command using `--deploy-android=False --android-asset-mode=apk_files`.
   c. Run the WAF package command using `--deploy-android=False --android-asset-mode=apk_files`.
   d. Upload the app that is created to Device Farm.
   e. Schedule the test run on Device Farm.

2. After a test run is scheduled, you can track its status in the **Device Farm Log** section of the Deployment Tool. Information from Device Farm is refreshed every 10 seconds.

![Device Farm Log](image)

**Note**
The Device Farm test fails on some devices if the device does not respond to input quickly enough. If this occurs, set `r.ShadersAsyncCompiling=1` in your `system_android_es3.cfg` file so that the main UI thread is blocked by shader loads and continues to respond.

3. To open a test run in the Device Farm console, right-click the run, and choose **View details in AWS console**.
4. To delete a test run, right-click the run, and choose **Delete**.

Create virtual reality projects in Lumberyard

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Lumberyard's virtual reality (p. 3286) system integrates the use of the Oculus Rift and HTC Vive head-mounted displays (HMD) on PC gaming systems. Before using these head-mounted displays, read each manufacturer's safety guide:

- Oculus Rift Health and Safety Warning
- HTC Vive Safety and Regulatory Guide

To activate Lumberyard's virtual reality support, add the appropriate Virtual Reality Gem(s) (p. 3249) in the Project Configurator and then build your project (p. 61). By enabling the appropriate Virtual Reality Gem(s), your project becomes capable of working with the supported virtual reality device(s), after some additional configuration.

Use console variables (CVARs) (p. 3251) to activate and modify configurable features of the virtual reality system, such as resolution and performance specifications.

You can use Script Canvas for gameplay scripting of the HMD device.

For information on Lua scripting functions for VR, see VR Lua Functions (p. 3260).

Topics
- Configuring your Project for Virtual Reality (p. 3249)
- Configuring Required Console Variables (p. 3251)
- Using the InstantVR Slice (p. 3252)
- Previewing your Virtual Reality Project (p. 3258)
- Debugging your Virtual Reality Project (p. 3258)
- Using EBus Request Bus Interface for Virtual Reality (p. 3259)
- VR Lua Functions (p. 3260)

Configuring your Project for Virtual Reality

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Add one or more Virtual Reality gems available in Lumberyard Editor to enable virtual reality for supported head-mounted displays (HMDs). You can add the gem(s) to new or existing projects. If you add more than one gem, the system automatically detects which HMD is connected, and uses the appropriate gem code to control the specific HMD and any associated virtual reality (VR) controllers.

Supported HMDs include:

- **Oculus** – Oculus Rift HMD
- **OpenVR** – HTC Vive HMD

**Note**

Use the NullVR Gem to run your level through the HMD framework without a connected VR device. This is useful for VR graphics debugging. For more information about this Gem, see NullVR Gem (p. 1180).

To add the Virtual Reality Gem(s)

1. Use the Lumberyard Setup Assistant to open the Project Configurator.
2. Select the project you want to add the Virtual Reality Gem to, or create a new project. Then click **Set as Default**.

3. Click **Enable Gems** below the project name.

4. Type **VR** into the search tool. Enable the **HMD Framework** Gem and one or more of the Virtual Reality gems:
   - **Oculus**
   - **OpenVR**
   - **NullVR**

5. Click **Save**.

6. After you enable the gem(s), you must build your project (p. 61) before they are available in Lumberyard Editor.
Configuring Required Console Variables

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Console variables – Console variables (CVARs) are a type of variable that you can manipulate in Lumberyard’s console interface. For more information, see Using the Console Window (p. 210).

You must set the following console variable to enable your project’s capability to support the head-mounted display.

`output_to_hmd = 1`

Enables output to head-mounted display (HMD). Allows users to toggle stereoscopic output while playing the game. With this variable enabled, the height and width resolution for the connected headset is detected and set automatically.

Set the following console variables to 0 in order to turn them off. These features are either unnecessary for virtual reality or too resource-intensive for a virtual reality environment.

`r_DepthOfField = 0`

Disables the depth of field setting. 0 = disabled; 1 = enabled; 2 = hdr time of day enabled.

`r_MotionBlur = 0`

Disables the motion blur setting. 0 = no motion blur; 1 = camera and object motion blur; 2 = debug mode.

`r_ResolutionScale`  
Float value. Scales the resolution for better performance. For example, set to 0.5 to scale the resolution by 50% in width and height (retains the aspect ratio).

`e_gi = 0`

Disables the global illumination setting. 0 = disabled; 1 = enabled.

Optional Console Variables

The following console variables are optional but strongly recommended. Disabling the following rendering features ensures better performance in a virtual reality environment. If you need certain rendering features that are explicitly disabled by these example variables, you may turn them back on at the cost of performance.

`sys_spec = 2`

Sets the system configuration specification to medium. 0 = custom; 1 = low; 2 = medium; 3 = high; 4 = very high.

`r_ssdoHalfRes = 3`

Applies screen space directional occlusion (SSDO) (p. 3285) bandwidth optimizations to half resolution output. 0 = full resolution; 1 = lower resolution; 2 = low res depth (except for small camera field of views) to avoid artifacts; 3 = half resolution output.
Using the InstantVR Slice

This topic references tools and features that are legacy. If you want to use legacy tools in Lumberyard Editor, disable the CryEntity Removal gem using the Project Configurator or the command line. To learn more about legacy features, see the Lumberyard Legacy Reference.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The instantVR slice is a set of entities, scripts, and assets that provide basic pieces of VR functionality to give you a starting point from which you can build your own VR application.

The VR functionality provided in this slice includes:

- Implementation of models-tracked controllers
- Teleportation using a navigation mesh to define a valid area
- Generation of a starting navigation area

The instantVR slice is part of the Virtual Reality Project sample level. Follow the instructions in VirtualRealityProject (p. 164) to download, install, and select the Virtual Reality Project before performing the following procedure.

To use the instantVR slice

1. In the Asset Browser (p. 296), navigate to dev\VirtualRealityProject\slices\.
2. Drag `instantvr.slice` into the viewport.

   The Lumberyard beaver is the starting location in this slice. Two controllers and a navigation area also appear.

3. Click the **VR Preview** button to enable VR preview, and then press **Ctrl+G** to run your level.

   You can use the trigger buttons on your controllers to teleport around the space.
InstantVR Lua Script Properties

You can view instantVR's assets in the Entity Outliner. The instantVR slice contains a Lua script called instantVR, which includes many of the child entities that add controller tracking and provide teleport support.
The `instantVR` Lua script has the following properties:

**TeleportInputEventNameRight**
- Name of the input event that triggers the right controller teleport.

**ControllerEntityRight**
- Name of the right controller entity.

**TeleportUseNavMesh**
- If selected, teleport function uses the navigation area for validation.

  **Note**
  If you select `TeleportUseNavMesh` without selecting `TeleportUseTerrain`, then teleport ray cast uses only the nav mesh to determine the valid area. If it finds no nav mesh, it displays an 'invalid' indicator on terrain.
  If both are selected, then teleport validation uses a ray cast to the terrain and to the navigation mesh. If both beams hit terrain and nav mesh, then the teleport location is valid, otherwise it is invalid.

**TeleportEntityInvalid**
- Name of the entity that spawns the invalid teleport location entity.

**TeleportEntityValid**
- Name of the entity that spawns the valid teleport location entity.

**TeleportInputEventNameLeft**
- Name of the input event that triggers the left controller teleport.

**TeleportMaxDistance**
- Maximum distance for teleporting.

**CameraEntity**
- Name of the camera entity, which determines the transform of the HMD and controllers.

**TeleportUseTerrain**
- If selected, the teleport function uses the terrain for validation.

  **Note**
  If you select `TeleportUseTerrain` without selecting `TeleportUseNavMesh`, then the teleport ray cast uses the terrain for validation only. Teleport will be valid as long as it is in contact with terrain.
  If both are selected, then teleport validation uses a ray cast to the terrain and to the navigation mesh. If both beams hit terrain and nav mesh, then the teleport location is valid, otherwise it is invalid.

**TeleportBeamSpawner**
- The name of the teleport beam arc entity along which to spawn entities, though this does not follow the ray cast line exactly.

To customize the `instantVR` Lua script, try the following modifications:

- Change the controllers to a different model by changing the *Static asset* of the *Static Mesh* component.
- Change the `TeleportMaxDistance` to a large value.
- Deselect `TeleportUseNavMesh` and climb the walls.
- Open the `InvalidLocationSpawner` and `ValidLocationSpawner` dynamic slices and change the entities they spawn.
InstantVR Known Issue

If you modify the size and position of the navigation area in the Lumberyard Editor, you must manually rebuild by clicking Game, AI, Generate Triangulation. Restarting the Lumberyard Editor or reloading the level also automatically rebuilds the navigation area.

Previewing your Virtual Reality Project

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can preview your virtual reality project for any project that has one or more of the virtual reality head-mounted display gems enabled. As you work in Lumberyard Editor, use your head-mounted display to preview your virtual reality game. The preview display inside of Lumberyard Editor is a preview only; it is not a good indicator of how fast the application will perform outside of the editor.

To gauge the game performance outside of Lumberyard Editor, create a release build (p. 344) to run your game in standalone mode.

To preview your virtual reality project

1. In Lumberyard Editor, click VR Preview on the bottom toolbar.

2. Enter game mode by doing one of the following:
   - Press Ctrl + G
   - On the main menu, click Game, Play Game.

To exit virtual reality preview mode

1. Exit game mode by pressing Esc.
2. Click VR Preview if you want to return to the default PC game preview mode.

Debugging your Virtual Reality Project

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can debug your virtual reality project either through a running instance of the game or through Lumberyard Editor. The head-mounted displays that Lumberyard supports outputs debugging information when debugging is enabled.

To enable debugging

- Enable one or both of the following console variable (p. 3251)s:
• Set `hmd_debug_info` to 1 (enabled) – Enables display of debug information provided by the associated HMD SDK.

• Set `hmd_debug_camera` to 1 (enabled) – Tests an editor-style debug camera at runtime. With this setting, players can use WASD keys to control the camera relative to the camera's facing direction, including the HMD. In regular (non-VR) mode, hold down the right mouse button to manipulate the camera's rotation.

When in debug mode, motion controllers appear as white crosshairs. That is, if you assigned an object or entity to represent the motion controller in the gameplay world, you will see it rendered with white crosshairs. The following picture shows two controllers, one with render geometry assigned, and the other without.

Using EBus Request Bus Interface for Virtual Reality

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Use the following request function with the StereoRendererBus event bus (EBus) interface to communicate with other components of your game.

For more information about using the EBus interface, see Working with the Event Bus (EBus) system (p. 1851).

```
IsRenderingToHMD
```

Returns true if the renderer is rendering to the HMD.
VR Lua Functions

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Lua bindings to interact programmatically with head-mounted display (HMD) devices that provide Virtual Reality (VR) experiences.

For general information on configuring your Lumberyard game project for VR, see Create virtual reality projects in Lumberyard (p. 3248).

Global Functions

The following functions provide programming interfaces for HMD devices.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMDDeviceRequestBusSender</td>
<td>Returns an object that is connected to the specified entity. For more information, see HMDDeviceRequestBus (p. 3260).</td>
</tr>
<tr>
<td>ControllerRequestBusSender</td>
<td>Returns an object that is connected to the specified entity. For more information, see ControllerRequestBus (p. 3261).</td>
</tr>
</tbody>
</table>

HMDDeviceRequestBus

Contains functions that return information about an HMD and control its pose and tracking level.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool IsInitialized()</td>
<td>Returns true if an HMD has successfully initialized on the bus. Returns false if no HMD is connected or failed to initialize.</td>
</tr>
<tr>
<td>Void RecenterPose()</td>
<td>Causes the direction that the HMD is currently facing to be considered forward'.</td>
</tr>
<tr>
<td>Void OutputHMDInfo()</td>
<td>Outputs the information about the currently connected HMD (contained in the HMDDeviceInfo object) to the console and log file.</td>
</tr>
</tbody>
</table>
### VR Lua Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Void SetTrackingLevel(int)</td>
<td>Sets the tracking level for the HMD. 0 specifies head level tracking (the player is standing); 1 is floor level tracking (the player is seated or on the floor).</td>
</tr>
<tr>
<td>HMDDeviceInfo GetDeviceInfo()</td>
<td>Returns an HMDDeviceInfo object that contains device information about a connected HMD. For more information, see struct HMDDeviceInfo (p. 3261).</td>
</tr>
<tr>
<td>TrackingState GetTrackingState()</td>
<td>Returns a TrackingState object that contains the most recent tracking information about a connected HMD. For more information, see struct TrackingState (p. 3261).</td>
</tr>
</tbody>
</table>

## ControllerRequestBus

Returns status information about an HMD controller.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool IsConnected(int controllerIndex)</td>
<td>Returns true if the given controller is connected, false if a controller is not connected. Pass 0 for the left controller, pass 1 for the right controller.</td>
</tr>
<tr>
<td>TrackingState GetTrackingState(int controllerIndex)</td>
<td>Returns a TrackingState object that contains tracking info about a connected controller. Pass 0 for the left controller, pass 1 for the right controller. For more information, see struct TrackingState (p. 3261).</td>
</tr>
</tbody>
</table>

## struct HMDDeviceInfo

Contains information about a device that displays on the screen when the device is detected.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String productName</td>
<td>Name of the connected HMD. The default is nullptr.</td>
</tr>
<tr>
<td>String manufacturer</td>
<td>Name of the company that manufactured the connected HMD. The default is nullptr.</td>
</tr>
<tr>
<td>Int renderWidth</td>
<td>The render width for the HMD in pixels. This is normally half the full resolution of the device (rendering is per eye). The default is 0.</td>
</tr>
<tr>
<td>Int renderHeight</td>
<td>The render height in pixels for a single eye of the HMD. The default is 0.</td>
</tr>
<tr>
<td>Float fovH</td>
<td>The horizontal field of view for both eyes in radians. The default is 0.0f.</td>
</tr>
<tr>
<td>Float fovV</td>
<td>The vertical field of view in radians. The default is 0.0f.</td>
</tr>
</tbody>
</table>

## struct TrackingState

Stores position and connection state information about the HMD. When an HMD is in use, certain parts of the device can go offline or online. For example, a controller can be disconnected, or the HMD can
temporarily lose rotational tracking. You can use the TrackingState to determine what part of the pose is currently valid.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoseState pose</td>
<td>The position and orientation in object space of the HMD. For more information, see struct PoseState (p. 3262).</td>
</tr>
<tr>
<td>DynamicsState dynamics</td>
<td>Contains the current state of the physics dynamics for the current device such as linear velocity, angular velocity, and acceleration. For more information, see struct DynamicsState (p. 3262).</td>
</tr>
<tr>
<td>Int statusFlags</td>
<td>Bit field that describes the current tracking state. For bit flags, see the enum HMDStatus (p. 3262).</td>
</tr>
</tbody>
</table>

**struct PoseState**

A specific pose of the HMD device. Each HMD device has its own way of representing its current pose in three dimensional space. This structure acts as a common data set between a connected device and the rest of the system. All data is in a local coordinate space.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quaternion orientation</td>
<td>A quaternion representing the current orientation in object space of the HMD.</td>
</tr>
<tr>
<td>Vector3 position</td>
<td>A three dimensional vector representing the current position of the HMD in object space as an offset from the centered pose.</td>
</tr>
</tbody>
</table>

**struct DynamicsState**

Dynamics (accelerations and velocities) of the current HMD. Many HMDs have the ability to track the current movements of VR devices for prediction. Not all devices support velocities and accelerations. All data is in a local coordinate space.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector3 angularVelocity</td>
<td>A three dimensional vector representing angular velocity in object space.</td>
</tr>
<tr>
<td>Vector3 angularAcceleration</td>
<td>A three dimensional vector representing angular acceleration in object space.</td>
</tr>
<tr>
<td>Vector3 linearVelocity</td>
<td>A three dimensional vector representing linear velocity in object space.</td>
</tr>
<tr>
<td>Vector3 linearAcceleration</td>
<td>A three dimensional vector representing linear acceleration in object space.</td>
</tr>
</tbody>
</table>

**enum HMDStatus**

The following code shows the status flags for HMDStatus.

```c
enum HMDStatus
{
```
Create macOS projects in Lumberyard

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

You can use Lumberyard to build macOS applications. Lumberyard includes four macOS-supported sample projects that you can use to learn how to build assets for macOS games using the Asset Processor, build shaders using the remote shader compiler, and build and deploy macOS applications using the Lumberyard build tools.

Topics
- Prerequisites (p. 3263)
- Setting Up Your Mac (p. 3263)
- Building macOS Games (p. 3264)
- Building Game Assets for macOS Games (p. 3265)
- Building Shaders for macOS Games (p. 3265)
- Running macOS Games (p. 3266)
- macOS Debugging and Troubleshooting (p. 3267)
- Creating a Project for Your macOS Games (p. 3268)

Prerequisites

To build games for macOS or iOS, Lumberyard requires the following on your Mac:

- Lumberyard Mac Support Files
- Xcode 11 or later
- macOS Mojave or later

Note
Lumberyard Editor requires Windows 10 to edit levels. You must have access to a PC with Lumberyard installed and be able to navigate and run commands from Terminal on your Mac.

Setting Up Your Mac

After you download and extract Lumberyard on your Mac, you must run Lumberyard Setup Assistant to install the third-party software that is required to run the game and compile the game code, engine, and asset pipeline.
To run Lumberyard Setup Assistant

1. Open the directory where you extracted Lumberyard and navigate to the /dev/Tools/LmbrSetup/Mac directory. Run the SetupAssistant.
2. Verify that the engine root path is correct.
3. On the Get started page, select the following and then click Next:
   - Create, Modify and Build projects
     Note
     Lumberyard Editor is not supported on macOS. Selecting this option enables the ability to build the asset processor and resource compiler only.
   - Compile for iOS devices
   - (Optional) Compile for Android devices
     Note
     Select this option if you are developing for Android devices. You must have the Android SDK installed on your Mac.
4. Follow the instructions onscreen to complete the installations for any third-party software or SDKs that you need. For more information about using Lumberyard Setup Assistant, see Using Lumberyard Setup Assistant to Set Up Your Development Environment (p. 16).
5. Open a command line window and navigate to your Lumberyard dev directory.
6. To initialize the build system, run the following command:
   ```
   sh lmbr_waf.sh configure
   ```
7. In the Finder, open the user_settings.options file (located in the /lumberyard/dev/_WAF_/ directory).
8. Verify that enabled_game_projects is set to your game project. For example, you can set this option to SamplesProject. If enabled_game_projects is not set correctly, edit and save the user_settings.options file and then run the configure command (sh lmbr_waf.sh configure) again.

Building macOS Games

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Before you can run your game on a macOS computer, you must ensure the shader compiler (located in the lumberyard_version\dev\Tools\CrySCompileServer\x64\profile directory) is running on your PC. For more information, see Building Shaders for macOS Games (p. 3265).

To build your game for macOS

1. On your Mac, in a Terminal window, navigate to the root directory of your Lumberyard installation (lumberyard_version/dev).
2. To generate an Xcode project and prepare the Lumberyard build system to build your app, run the following command:
   ```
   sh lmbr_waf.sh configure xcode_mac
   ```
   Note
   Metal is the default renderer.
3. Do one of the following to build your game:
Building Game Assets for macOS Games

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

When you build a macOS game using Lumberyard, you must first build the assets that are included with the application. All built assets are located in the cache folder of your Lumberyard installation. For example, when you build the Samples Project, the assets are saved to the lumberyard_version\dev \cache\SamplesProject\osx_gl directory. The initial build of the Samples Project assets may take up to an hour to process, but incremental changes should process almost instantly.

Note
If you make changes to your game in Lumberyard Editor on your PC, you must copy the updated assets to your Mac using your preferred method. For example, you can use source control. For information, see Using the Perforce Plugin with Lumberyard (p. 23).

To build macOS game assets on your Mac
1. On your Mac, close all instances of the Asset Processor.
2. Edit the bootstrap.cfg file (located in the lumberyard_version/dev directory) to set sys_game_folder to SamplesProject (or the project you want to build). Save the file.
3. Edit the AssetProcessorPlatformConfig.ini file (located in the lumberyard_version/dev directory) to uncomment osx_gl=enabled (remove the preceding semicolon) and to comment out pc=enabled (add a preceding semicolon). Save the file.
4. In a command line window, navigate to the lumberyard_version/dev/BinMac64 directory and run Asset Processor (GUI or batch version) to process and build your game assets.

Building Shaders for macOS Games

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.
Lumberyard uses a versatile shader system to achieve high quality, realistic graphics. Because the shader compilation pipeline depends on the Windows-specific HLSL optimizer, you must connect to a shader compiler on your PC when running a game on macOS during development. This compiles the subset of shaders required by your game, on demand.

**Note**
You must connect your PC and macOS computer to the same network and configure any firewalls to allow traffic through port 61453.

When a new shader is compiled, the game waits for the binary shader permutation to compile on your PC and be sent back to your macOS computer. Once this occurs, the shader is cached locally. When you are ready to release your game, you must pack up and include all cached binary shaders.

You can use an allow list to specify the IP addresses that are allowed to connect to your remote shader compiler. For information, see Creating an allow list for the Remote Shader Compiler (p. 1647).

**To build the shader compiler (if not already done)**

1. On your PC, in a command line window, navigate to the `lumberyard_version\dev` directory.
2. Enter the build command for your version of Visual Studio:

   ```
   lmbr_waf build_win_x64_vs2017_profile -p all --targets=CrySCompileServer
   ```

**To run the shader compiler on your PC**

1. Navigate to the `lumberyard_version\dev` directory.
2. Edit the `system_osx_osx_gx.cfg` file to set the `localhost` for `r_ShaderCompilerServer` to the IP address of the PC on which you will run the shader compiler.
3. Launch Asset Processor if it is not still running.
4. Verify that you are sharing the `cache` folder between your PC and Mac by checking the corresponding cache file (located in the `lumberyard_version\dev\cache\SamplesProject \ios\system_osx_osx_gx.cfg` directory).

**Running macOS Games**

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Before you can run your game on a macOS computer, you must ensure the shader compiler (located in the `lumberyard_version\dev\Tools\CrySCompileServer\x64\profile` directory) is running on your PC. For more information, see Building Shaders for macOS Games (p. 3265).

**To run your game on a macOS computer**

1. Open the Xcode solution that you generated (located in the `Solutions` folder in the directory where you installed Lumberyard).
2. Build, run, and debug your application as you would any Xcode project. For information, see Launching Your Mac App.
3. (Optional) Load different levels by doing one of the following:

- Open the console window using the tilde (~) key, and then enter `map <name of map or level to load>`. The console supports tab completion, so you can press Tab after the map command to see the list of supported levels and maps to load.

- Edit the `autoexec.cfg` file for your game to change the map command to load a different map or level. Run the game from Xcode again.

  For example, if you are using the Samples Project, edit the `autoexec.cfg` file located in the `/dev/SamplesProject` directory. macOS supports the Advanced_RinLocomotion level.

4. Use the following controls to navigate around your game:

- Switch between cameras by selecting the buttons in the lower right corner of the screen.
- Move Rin in the Character Controller view by using the mouse or keyboard (WASD).
- Jump in the Character Controller view by pressing the Space key.

---

macOS Debugging and Troubleshooting

This feature is in preview release and is subject to change.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard provides full access to the source code, which allows you to debug your macOS application using Xcode without additional Lumberyard-specific steps to follow. For information about debugging and profiling your macOS application, see Debugging in the official Apple developer documentation.

Unable to see activity in the shader compiler window

You must connect to the shader compiler on your PC in order to compile the subset of shaders required by your game, on demand. To verify that your app has connected correctly and obtained all shaders, you can view the output in the shader compiler window. If you still do not see any activity in the window, please check your setup by following the instructions on the Building Shaders for macOS Games (p. 3265) page.

Cleaning the project does not create a full rebuild of the macOS application

Lumberyard uses a custom build step to generate the final executable and temporary C++ object files, which output to the \BinTemp\darwin_x64_debug or \BinTemp\darwin_x64_profile directory where you installed Lumberyard. Unlike a regular Xcode project, in order to create a full rebuild of the macOS application, you must manually delete the contents of the output folder or run one of the following Waf commands from a Terminal window:

- To build debug, run the following command: lmbr_waf.sh clean_darwin_x64_debug
- To build profile, run the following command: lmbr_waf.sh clean_darwin_x64_profile
- To build release, run the following command: lmbr_waf.sh clean_darwin_x64_release

Observed frame rate varies greatly

While running your application, the observable frame rate can vary depending on the build (debug or profile) you are running and whether you are connected to the Xcode debugger. To display the frame rate in the upper right corner of the screen, set the r_DisplayInfo configuration variable to 1 or higher. When your Xcode project is generated, the default build scheme is set up for debugging. If you want to test or profile your application’s speed, we recommend that you edit your active scheme to run a profile build. Deselect Debug executable.

Creating a Project for Your macOS Games

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The topics in Create macOS projects in Lumberyard (p. 3263) demonstrate how to use the Samples Project that is included with Lumberyard to build game assets, shaders, and macOS applications. You can follow the same instructions to create a project for your own macOS game.

Note

Ensure you have the prerequisites (see Create macOS projects in Lumberyard (p. 3263)) and your Mac is properly set up to compile for macOS computers.
To create a project for your macOS game

1. On your PC, use the Project Configurator to create a new project. For information, see Creating Lumberyard projects (p. 43).
2. Submit the new project into your revision control system and then check out the project onto your Mac.
3. Edit the user_settings.options file (located in the lumberyard_version\dev\_WAF_ directory) to set enabled_game_projects to the name of the project you created:

   ```
   [Game Projects]
   enabled_game_projects = MyProject
   ```

   You can simultaneously build multiple projects by separating each project name with a comma:

   ```
   [Game Projects]
   enabled_game_projects = SamplesProject,MyProject,OtherProject
   ```

4. In a command line window, configure and build your project using the instructions on the Building macOS Games (p. 3264) and Running macOS Games (p. 3266) pages.

   **Note**
   If you enabled multiple projects, you can switch between multiple targets in your Xcode project.

Creating Lumberyard Executables for Linux

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard supports compiling a Windows client for a multiplayer project and connecting to a Linux dedicated server. You must do the following to ensure that the Windows client works properly on a Linux dedicated server:

- Compile the assets on a Windows computer (p. 344)
- Compile the server executable for use on a Linux computer (p. 3270)
- Compile a Windows client to use to connect to the Linux server (p. 61)

When these tasks are complete, you can deploy assets to either a Linux server or Windows client.

Topics

- Prerequisites (p. 3270)
- Deploy a Lumberyard multiplayer project's server on Linux (p. 3270)
- Testing the Windows Client to Linux Server Connection (p. 3272)
- Preparing the Amazon GameLift Package (p. 3274)
Prerequisites

To create Lumberyard executables for Linux, you must have the following:

- A Windows 10 Lumberyard install configured to compile your project and the Lumberyard engine.
- A computer running Ubuntu Bionic (18.04 LTS) where you have superuser access, with at least 50Gb of available hard drive space. This can be a native host, Windows Subsystem for Linux (WSL), or a virtual machine. For instructions on hosting an Ubuntu instance on Amazon Elastic Compute Cloud, see Getting started with Amazon EC2 Linux Instances. When you create an Amazon EC2 instance, pick Ubuntu 18.04 LTS as the image and x86_64 as the architecture to ensure that Lumberyard is supported.

Important

In Lumberyard version 1.23 and later, if you enable the EMotionFX Gem on Linux, then you must install libxmu to run the Linux Dedicated Server. Use the following command to install libxmu:

```
apt install libxmu6
```

Deploy a Lumberyard multiplayer project's server on Linux

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

To distribute your Lumberyard project's server onto Linux, you need to have access to a Windows 10 computer to perform your first build, which generates the client executable and assets that the server uses. After your build is complete, bundle up the assets and code for distribution, building, and hosting on your Linux server. This topic walks you through the steps to perform the Windows 10 client build, get the files onto your Linux host, and then build the Linux server application.

Currently, the only officially supported Linux distribution is Ubuntu 18.04 LTS. To verify that you're running the correct distribution on your Linux host, run the `lsb_release -a` command. If you're on the correct distribution, you should see Ubuntu 18.04 in the output.

These instructions are for building the MultiplayerSample project that's included in the Lumberyard distribution. When you build and distribute your project, edit the scripts referenced in this topic where needed to reference your project's name and contents instead.

Bundle the assets and source on Windows

To create a Linux server for your project, you first have to build the assets on Windows and bundle them for distribution to the server. This distribution also includes all of the source code and tools that you need to generate a server build, so you don't need to install Lumberyard on your Linux host.

To package the assets and source on Windows

1. Open a console and navigate to the `lumberyard_install\dev` directory.
2. Open `bootstrap.cfg` and set the value of `sys_game_folder` to MultiplayerSample.
3. Build the project assets for the multiplayer server.

```bash
sys_game_folder=MultiplayerSample
```

4. Bundle the packaged assets together into a tape archive (.tar) file for distribution to a Linux host. This requires Python, which is distributed with Lumberyard:

```bash
Tools\Python\version\windows\python.exe Tools\LmbrSetup\Linux\archiver.py
```

This tool generates a file based on the current timestamp, located at `BinTemp\unix_archives\YYYY-MM-DD_HH-mm-ss.tar`

**Note**
This archive file is uncompressed and is quite large. Before distributing it to a remote Linux host, you might want to compress it with LZMA, bzip2, or gzip.

5. Copy the created archive over to your Linux host. You can use SFTP, SCP, or any other method that you would normally use to achieve this. For instructions on copying to an Amazon EC2 instance using SCP, see Transfer files to your Linux instance using WinSCP.

If you're running Windows 10, you can use the Windows Subsystem for Linux (WSL) to test a deployment. See Microsoft's instructions on installing WSL and initializing a distribution to get set up with WSL. You can access the file directly from the WSL mount points for your Windows drives. The drives are available from within a WSL terminal at `/mnt/drive_letter`. To avoid performance problems when running the dedicated server under WSL, copy the archive into WSL's file system before continuing.

### Compile the dedicated server on Linux

1. When the packaged assets and source are copied to your Linux host, extract them so that you can build the dedicated server.

```bash
cd your_upload_path
tar -xvf YYYY-MM-DD_HH-mm-ss.tar
```

**Note**
If you compressed your archive in an earlier step, add a decompression argument to `tar`:

- Compressed with LZMA: `tar -xJvf YYYY-MM-DD_HH-mm-ss.tar.xz`
- Compressed with bzip2: `tar -xyvf YYYY-MM-DD_HH-mm-ss.tar.bz2`
- Compressed with gzip: `tar -xzvf YYYY-MM-DD_HH-mm-ss.tar.gz`

2. With the archive unpacked, change to the project's `dev` directory:

```bash
cd MultiplayerSample/dev
```

3. Check that your Linux host has the correct dependencies installed and update them if needed:

```bash
sudo ./Tools/LmbrSetup/Linux/setup.sh
```

While setting up your environment, this script might prompt you for input to confirm adding a package repository (PPA) or install a package. If you need to automate server setup, edit this script to make sure there are no user prompts.
4. Configure the Lumberyard build environment that's bundled as part of the distribution:

```
./lmbr_waf.sh configure --3rdpartypath absolute_path_to_unarchive_location/
MultiplayerSample/3rdParty/ 
--bootstrap-tool-param "--enablecapability compileengine --enablecapability
compilegame" 
--update-settings True
```

5. Edit the `MultiplayerSample_pc_Paks_Dedicated/system_linux_pc.cfg` file. Set the `log_RemoteConsoleAllowedAddresses` value to a comma-separated list of the IP addresses that Windows clients will connect from.

6. (For non-release builds only) Build the builder assistant tool binaries.

```
./lmbr_waf.sh --3rdpartypath absolute_path_to_unarchive_location/3rdParty/
build_linux_x64_profile -p host_tools
```

7. Build the dedicated server:

```
./lmbr_waf.sh --3rdpartypath absolute_path_to_unarchive_location/
MultiplayerSample/3rdParty/ 
build_linux_x64_profile_dedicated -p game_and_engine
```

**Note**
To create a different type of build, such as a debug or release, change `profile` to the appropriate build type. For all of the available Linux server build types, see Waf Commands and Options (p. 90).

# Testing the Windows Client to Linux Server Connection

This feature is in preview release and is subject to change.

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

After you've built the Windows client and Linux server (p. 3270), you should test the connection to make sure that everything is running smoothly. This topic covers starting your server, getting connection information, running the remote console on Windows to manage your connection, and connecting your game client to the server.

**Prerequisites**
To test your server setup, complete the following tasks.

- Complete the steps in Deploy a Lumberyard multiplayer project's server on Linux (p. 3270).
- Get the public IP address (not DNS name) of your Linux server.
- Get the public IP address of your client machine.
- Allow outbound TCP connections on Windows 10 to your Linux server on port 4600, and outbound UDP connections on port 33435. If you're not familiar with how to configure Windows firewall to allow outbound connections, see the Microsoft Windows Firewall documentation.
• Allow inbound TCP connections on Linux on port 4600, and inbound UDP connections on port 33435. If you’re using an Amazon EC2 instance, follow the instructions for Authorizing inbound traffic on Linux. If you’re running your own independent Linux server or using Windows Subsystem for Linux (WSL), read your distribution’s instructions on configuring your firewall to allow the client connection.

**Important**
There might be other considerations in your network environment that require additional setup – such as a corporate firewall, proxy, or VPN – that you should consult with your system administrator on as part of setting up for testing.

## Start the server

Before starting the client and after configuring the server to accept connections, start the dedicated server.

### To start the server

1. In a terminal, navigate to the `absolute_path_to_unarchive_location/dev/BinLinux64.Dedicated` directory.
2. Copy the resources built on Windows to the build directory. To function properly, the working directory that you launch the server from needs to contain configuration information for the project and the `multiplayersample` directory containing the `.pak` asset bundles.

   ```bash
   cp -r ../MultiplayerSample_pc_Paks_Dedicated/* ./
   ```

3. Make sure that the correct project is bootstrapped. Edit the `absolute_path_to_unarchive_location/dev/BinLinux64.Dedicated/bootstrap.cfg` file to ensure that the value of `sys_game_folder` is set to `MultiplayerSample`.

   ```
   sys_game_folder=MultiplayerSample
   ```

   **Note**
   You might see `^M` characters at the end of every line in the `bootstrap.cfg` file. This indicates that the file uses Windows-style line endings rather than Linux-style ones. This doesn't affect the ability of the Lumberyard dedicated server from loading the bootstrap file correctly, and you don't need to worry about preserving the line ending type when you edit the file.

4. Start the server.

   ```
   ./MultiplayerSampleLauncher_Server
   ```

## Configure the server remotely

When the server is running, launch the remote console and set up the server to start hosting a game session.

1. Launch the Remote Console on your Windows 10 machine, located at `lumberyard_install_dir\dev\Tools\RemoteConsole\RemoteConsole.exe`.
2. In the Remote Console, Select `Targets > PC`, and then edit the `Targets > Custom IP` setting to be the IP address for your Linux server.
If the connection to the dedicated server is successful, you see a green status light and the word **Connected** in the lower right corner of the window. Make sure that the IP address and port, located in the lower left of the window, match up with the location of your server. If you had an instance of the client already running, the remote console will attempt to connect to it first.

3. In the command line located at the bottom of the window, send commands to the server to start the multiplayer session and load a map.

   ```plaintext
   mphost
   map multiplayersample
   ```

4. Close the remote console.

### Connect from the client to the server

Start the game client and connect to the multiplayer session.

1. Edit the `lumberyard_install_dir\dev\bootstrap.cfg` file and set `connect_to_remote` to `1`.

   ```plaintext
   connect_to_remote=1
   ```

2. Launch the client.

   - For Visual Studio 2017: `lumberyard_install_dir\dev\Bin64vc141\`
   - For Visual Studio 2019: `lumberyard_install_dir\dev\Bin64vc142\`

3. In the client, press the `\` (grave) key to open the console and enter the command `mpjoin server_ip_address`.

If the connection to the game server was successful, you now have a game session running in your client. Try out connecting other clients to the server using these same steps to continue testing with multiple players!

### Preparing the Amazon GameLift Package

This feature is in **preview** release and is subject to change.
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

Lumberyard has support for bundling and distributing your dedicated Linux servers over Amazon GameLift. To integrate your Lumberyard game with GameLift, enable the GameLift Gem (p. 1130) and use the GameLift C++ SDK API to configure your game server and client to manage server sessions and connections from a client application.

In the rest of this topic, you learn how to set up for creating your GameLift package and make a basic deployment.

Prerequisites

Before packaging and pushing your GameLift server, make sure that you complete the following.

- Set up your AWS account to use the GameLift service. Follow the GameLift - Setting Up instructions.
- Install the AWS CLI on your Linux server. Follow the AWS CLI install instructions for Linux.
- Build and do a test deploy of your game server on a standalone Linux machine (p. 3270).
- (Optional) Test client/server connectivity (p. 3272).

Prepare the GameLift package

The following instructions tell you how to package the MultiplayerSample project for a test deployment to GameLift. When you’re ready to deploy your game, edit the MultiplayerSample_CreateGameLiftPackage.sh script to use your project and its asset bundles.

1. On your Linux server, run the packaging script.

   ```bash
   ./MultiplayerSample_CreateGameLiftPackage.sh
   ```

2. Push your GameLift package to AWS using the AWS CLI.

   ```bash
   aws gamelift upload-build --operating-system AMAZON_LINUX \
   --build-root "./GameLiftPackageLinux" \
   --name "your package name" \
   --build-version "your build version" \
   --region us-west-2
   ```

   **Note**
   If you want your servers to be hosted in a different AWS Region, replace the us-west-2 argument value with the appropriate region identifier.

Next steps

Now that you have your server package uploaded to AWS, take these next steps to get your game deployed over GameLift and do more advanced configuration.

- Set up a deployment fleet
- Configure region-based queuing
- Examine metrics and data
- Secure your GameLift servers
Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

The Amazon Lumberyard engine, integrated development environment, and related assets and tools are licensed as "Lumberyard Materials" under the terms and conditions of the AWS Customer Agreement and the Lumberyard Service Terms. Please see these terms and conditions for details.

Topics
- Lumberyard Redistributables (p. 3276)
- Alternate Web Services (p. 3278)

Lumberyard Redistributables

For purposes of the Lumberyard Service Terms, the Lumberyard materials in the directories listed below are designated as "Lumberyard Redistributables." Unless subdirectories of a directory are specified, all files in the directory listed are deemed Lumberyard Redistributables.

Note
Restrictions on use and distribution of the Lumberyard materials, including in source code form, are specified in the Service Terms.

Lumberyard

- \3rdParty\GameLift
- \dev\WAF_
- \dev\Bin64
- \dev\CloudGemSamples
- \dev\Code\CloudGemSamples
- \dev\Code\CryEngine
- \dev\Code\Framework
- \dev\Code\Launcher
- \dev\Code\MultiplayerProject
- \dev\Code\SamplesProject
- \dev\Code\Sandbox
- \dev\Code\Tools
- \dev\Code\Tools\AssetTagging
- \dev\Code\Tools\ClangReflect
- \dev\Code\Tools\CryCommonTools

Version 1.28
3276
Lumberyard Redistributables

- \dev\Code\Tools\CryD3DCompilerStub
- \dev\Code\Tools\CrySCompilerServer
- \dev\Code\Tools\CryXML
- \dev\Code\Tools\DBAPI
- \dev\Code\Tools\GemRegistry
- \dev\Code\Tools\HLSLCrossCompiler
- \dev\Code\Tools\LUARemoteDebugger
- \dev\Code\Tools\PRT
- \dev\Code\Tools\RC
- \dev\Code\Tools\ShaderCacheGen
- \dev\Code\Tools\SphericalHarmonics
- \dev\Code\Tools\AssetProcessor
- \dev\Editor
- \dev\Editor
- \dev\Editor
- \dev\Gems
- \dev\MultiplayerProject
- \dev\ProjectTemplates
- \dev\SamplesProject
- \dev\Tools\Build\waf-1.7.13
- \dev\Tools\lmb_aws\AWSResourceManager\default-project-content
- \dev\AssetProcessorPlatformConfig.ini
- \dev\bootstrap.cfg
- \dev\editor.cfg
- \dev\engineroot.txt
- \dev\lmb_aws.cmd
- \dev\lmb_waf.bat
- \dev\lmb_waf.exe
- \dev\SetupAssistantConfig.json
- \dev\system_BuildShaderPak_DX11.cfg
- \dev\system_BuildShaderPak_GL4.cfg
- \dev\system_windows_pc.cfg
- \dev\waf_branch_spec.py
- \dev\wscript

Asset Collection – Woodland
- All directories

Asset Collection – Beach City
- All directories

Legacy Sample (GameSDK)
- All directories
Alternate Web Services

Open 3D Engine (O3DE), the successor to Lumberyard, is now available in Developer Preview. Download O3DE or visit the AWS Game Tech blog to learn more.

For purposes of the Lumberyard Service Terms, "Alternate Web Service" means any non-AWS compute, database, storage, or container service that is similar to or can act as a replacement for the following services: Amazon EC2, Amazon Lambda, Amazon DynamoDB, Amazon RDS, Amazon S3, Amazon EBS, Amazon EC2 Container Service, or Amazon GameLift.
Lumberyard documentation is available in PDF format for the following versions.

<table>
<thead>
<tr>
<th>Lumberyard Version</th>
<th>Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-02-09 (1.0)</td>
<td>Getting Started Guide</td>
</tr>
<tr>
<td>2016-03-10 (1.1)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2016-04-26 (1.2)</td>
<td>Getting Started Guide</td>
</tr>
<tr>
<td>2016-08-08 (1.4)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2016-09-22 (1.5)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2016-11-03 (1.6)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2017-02-22 (1.8)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2017-08-01 (1.10)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2017-12-14 (1.12)</td>
<td>Getting Started Guide   User Guide   Developer Guide</td>
</tr>
<tr>
<td>2018-11-13 (1.16)</td>
<td>Getting Started Guide   User Guide</td>
</tr>
<tr>
<td>2018-12-18 (1.17)</td>
<td>Getting Started Guide   User Guide</td>
</tr>
<tr>
<td>2019-03-14 (1.18)</td>
<td>Getting Started Guide   User Guide</td>
</tr>
<tr>
<td>2019-09-12 (1.20)</td>
<td>Getting Started Guide   User Guide</td>
</tr>
<tr>
<td>2019-12-13 (1.21)</td>
<td>Getting Started Guide   User Guide</td>
</tr>
<tr>
<td>2020-02-14 (1.22)</td>
<td>Getting Started Guide   User Guide</td>
</tr>
<tr>
<td>Lumberyard Version</td>
<td>Guide</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>2020-06-25 (1.24)</td>
<td>Welcome Guide</td>
</tr>
<tr>
<td></td>
<td>User Guide</td>
</tr>
<tr>
<td>2020-10-01 (1.25)</td>
<td>Welcome Guide</td>
</tr>
<tr>
<td></td>
<td>User Guide</td>
</tr>
<tr>
<td>2020-12-04 (1.26)</td>
<td>Welcome Guide</td>
</tr>
<tr>
<td></td>
<td>User Guide</td>
</tr>
<tr>
<td></td>
<td>Tools UI Extensions Guide</td>
</tr>
<tr>
<td>2021-05-13 (1.27)</td>
<td>Welcome Guide</td>
</tr>
<tr>
<td></td>
<td>User Guide</td>
</tr>
<tr>
<td></td>
<td>Tools UI Extensions Guide</td>
</tr>
</tbody>
</table>
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>actor</td>
<td>A specialized entity (p. 3283) that is the basis for characters in a game.</td>
</tr>
<tr>
<td>additive animation</td>
<td>An animation that can be attached to a base animation to extend its behavior.</td>
</tr>
<tr>
<td>agent</td>
<td>An autonomous entity used in artificial intelligence (AI) that uses sensors to observe its environment and directs its activity towards achieving one or more goals.</td>
</tr>
<tr>
<td>aim pose</td>
<td>Part of a collection of parametric-blended poses for making a character take aim at specified points in the game.</td>
</tr>
<tr>
<td>alpha channel</td>
<td>An extension of RGB color values for specifying the opacity of an object. A value of 0.0 indicates fully transparent while a value of 1.0 indicates fully opaque.</td>
</tr>
<tr>
<td>Amazon GameLift</td>
<td>A fully managed AWS (p. 3281) service for deploying, operating, and scaling session-based multiplayer game servers in the cloud.</td>
</tr>
<tr>
<td>archetype entity</td>
<td>A special type of entity (p. 3283) with linked instances. If a parameter of the archetype entity is changed, all other instances of that entity parameter are automatically updated.</td>
</tr>
<tr>
<td>asset</td>
<td>Any art, texture, 3D model, sound effect, or other digital data that is presented to the user in the game.</td>
</tr>
<tr>
<td>attachment</td>
<td>A hierarchical object that is attached to characters, respond to real-world physics, and can be attached, detached, or replaced at runtime in the game. Character attachments include clothing, weapons, tools, or entire body parts such as heads or hands.</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services, an infrastructure web services platform in the cloud for companies of all sizes. See Also <a href="http://aws.amazon.com">http://aws.amazon.com</a>.</td>
</tr>
<tr>
<td>baked</td>
<td>Performs and stores all calculations for a scene element so that the element does not need to be processed or rendered in real time in the game. Often used for lighting or physics. Also referred to as prebaked.</td>
</tr>
<tr>
<td>bind pose</td>
<td>The pose that a character has when you bind the mesh (skin) to the skeleton. The skeleton determines the pose.</td>
</tr>
<tr>
<td>blend shape</td>
<td>Method that stores a deformed version of a mesh as a series of vertex positions. In each keyframe of an animation, the vertices are interpolated between these stored positions. Also known as morph target animation or per-vertex animation.</td>
</tr>
</tbody>
</table>
blend space | Animation blending that is treated as geometry. A character’s kinematic, physical, and other high-level motion-related parameters are mapped onto corresponding features that are stored in animation clips. By storing such motion as parameters, controllable interactive animations are possible. Specifically, an animation is associated with a 1D, 2D, or 3D location in the blend space. Also known as a bspace.

bloom | Effect that reproduces an imaging artifact of real-world cameras. The effect appears as fringes or feathers of illumination bleeding from the border of a bright area in an image, which gives the illusion of a bright light overwhelming the camera.

boids | Entities that mimic living animals and that have simulated group behavior and obstacle avoidance.

brush | A simple 3D shape that is tied to an entity, and that provides a specific appearance. Brushes are used for static objects.

bspace | See blend space.

bump map | A grayscale image that allows more realistic rendering of an object by introducing small displacements of its surface without changing its geometry. This is done by perturbing the surface normals of a rendered object during lighting. The amount of perturbation is specified by the values in the bump map.

Cloud Canvas | A tool for building connected gameplay by using Lumberyard and AWS services, such as Amazon Cognito, Amazon DynamoDB, AWS Lambda, Amazon S3, Amazon SNS, and Amazon SQS.

collision proxy | A simplified geometric shape for approximating a more complex piece of geometry for purposes of a fast first-pass collision detection.

cooking | Pre-processing step that converts data such as mesh geometry into formats that PhysX can use. Cooking also prepares data structures and computes intermediate results that can be performed offline, which saves computation at runtime.

cubemap | A set of six squares that represent reflections from the environment. The six squares form the faces of an imaginary cube that surrounds an object.

cutscene | A noninteractive cinematic game sequence that is typically used to promote plot during gameplay.

damping | The gradual reduction of movement, vibration, or intensity.

DCC | Digital content creation; related to a third-party product such as Autodesk 3ds Max or Autodesk Maya for creating digital assets.

decal | A 2D texture placed on a piece of flat geometry.

detail map | An image for adding up-close surface details to an object.

diffuse map | An image for defining the base color and pattern of an object's surface.

displacement map | A type of heightmap (p. 3283) that modifies the position of vertices of a surface by a specified amount.

DOF | Depth of field. The degree to which distant objects are in focus relative to closer ones.

EBus | A modular message dispatch system that enables components, entities, and other types of objects to communicate with one another with few or no
interdependencies. Because Ebuses are decoupled from each other, you can more easily build high performing and modular game systems. Also known as event bus.

emitter
An entity that specifies the location from which particles are emitted.

entity
A game object with one or more components that provide some behavior or functionality. An entity consists of a unique ID and a container.

environment probe
A technique that uses cube maps to provide a game level or location with realistic ambient lighting.

experimental
A designation for a Lumberyard tool that we recommend that you not use in production yet, as it is still in the early stages of development and we have no current plans to support it. However, it also means that feature is stable enough and functional for specific use cases. You may find the experimental feature helpful as a guideline for implementing your own game-specific features. API operations are subject to change.

gem
A package that contains code and assets to provide a single feature or multiple tightly scoped functions.

gloss map
An image that represents the microscale roughness of a surface. The gloss map is located in the alpha channel of the normal map.

heightmap
A grayscale image used to modify vertex positions of a surface. Lumberyard uses heightmaps to store terrain surface height data. White areas represent the high areas while black areas represent the low areas of the terrain.

HDR tone mapping
The process of converting the tonal values of an image from a high dynamic range (HDR) to a lower range.

helper
Visual icons attached to objects in Lumberyard Editor that provide object-specific functionality.

IK
Inverse kinematics. The use of kinematics equations to calculate the positions and orientations of joints of a character's skeleton so that a specific part of the skeleton (the end effector) reaches a defined target point.

IBL
Image-based lighting. A rendering technique that involves capturing lighting information, storing it in an environment probe, and projecting it onto a scene.

imposter
Procedurally created 2D sprites that are rendered to look like 3D objects. In essence, imposters are 2.5D objects.

k-means clustering
Method for partitioning a set of data points into k separate groups. Each group has a mean calculated from the average of the points it contains and each point is assigned to a group based on which group mean the point is closest to. The method usually works iteratively, alternating between updating point assignments and updating mean values until a stable partition is reached.

keyframe
An animation frame that specifies exact positions and orientations of geometry affected by the animation. Animation frames that exist between keyframes are interpolated based on animation curves.

legacy
A designation for Lumberyard tools that are no longer being advanced and will eventually be removed.

level
A world or map that represents the space or area available to the player during the course of completing a discrete game objective. Most games consist of multiple levels.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>locomotion locator</td>
<td>The Y vector of the character root joint quaternion, which is typically the direction in which the character is facing. The locomotion locator is needed for motions that translate in nonuniform ways, such as stop or start transitions that have changes in acceleration.</td>
</tr>
<tr>
<td>LOD</td>
<td>Level of detail. A technique for increasing performance and reducing draw calls by displaying progressively less-detailed objects the farther they are from the camera.</td>
</tr>
<tr>
<td>look pose</td>
<td>Part of a collection of parametric-blended poses for making a character look at specified points in the game.</td>
</tr>
<tr>
<td>mesh</td>
<td>A collection of vertices that define the surface of an object.</td>
</tr>
<tr>
<td>minimap</td>
<td>A miniature map placed at a screen corner in the game to aid players in orienting themselves in the world.</td>
</tr>
<tr>
<td>mip map</td>
<td>A precalculated, optimized sequence of textures, each of which is a progressively lower resolution representation of the same image. Used in conjunction with LOD (p. 3284) processing.</td>
</tr>
<tr>
<td>morph target</td>
<td>A snapshot of vertex locations for a specific mesh that have been deformed in some way.</td>
</tr>
<tr>
<td>morph target animation</td>
<td>See blend shape.</td>
</tr>
<tr>
<td>navmesh</td>
<td>A navigation mesh, or navmesh, defines the areas of an environment in which a character can move freely without obstructions such as trees, lavas, or other environmental barriers.</td>
</tr>
<tr>
<td>normal</td>
<td>The vector that is orthogonal to a surface defined by a set of vertices.</td>
</tr>
<tr>
<td>normal map</td>
<td>An image whose pixel values are interpreted as the normal vectors for each point on the surface to which the image is mapped.</td>
</tr>
<tr>
<td>null bone</td>
<td>The character bone associated with a null or root object.</td>
</tr>
<tr>
<td>parallax mapping</td>
<td>A technique that is used to create detail in a texture adding the illusion of depth. This depth perception changes based on perspective.</td>
</tr>
<tr>
<td>PBR</td>
<td>Physically based rendering. PBR uses real-world physical rules and properties to define how light interacts with the surface of objects. Used by the Lumberyard rendering system.</td>
</tr>
<tr>
<td>per-vertex animation</td>
<td>See blend shape.</td>
</tr>
<tr>
<td>POM</td>
<td>Parallax occlusion mapping. POM uses a displacement map to encode surface detail information in a texture. In this way self-occlusion and self-shadowing of an object is possible without changing the surface geometry.</td>
</tr>
<tr>
<td>prebaked</td>
<td>See baked.</td>
</tr>
<tr>
<td>prefab</td>
<td>A game object template that stores an asset or a group of assets and all associated properties.</td>
</tr>
<tr>
<td>Preview</td>
<td>A designation for Lumberyard tools that may be missing key features but are still stable and usable. The user experience is high quality, functional, and consistent where it exists but may be unfinished. APIs are subject to change.</td>
</tr>
<tr>
<td>procedural vegetation</td>
<td>A technique used to automatically cover a large area of terrain with vegetation objects using texture layers.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>project</td>
<td>The collection of levels, assets, and code that make up a game.</td>
</tr>
<tr>
<td>quaternion</td>
<td>Mathematical notation that represents the orientation and rotation of objects in three dimensions.</td>
</tr>
<tr>
<td>ragdoll</td>
<td>Physical rules used to simulate the realistic movement of a skeletal character.</td>
</tr>
<tr>
<td>rigging</td>
<td>The process of building a skeleton hierarchy of bone joints for a character mesh.</td>
</tr>
<tr>
<td>rope</td>
<td>Used for attaching cloth, hair, or ropes to a character so that the objects can dangle and move realistically against the character.</td>
</tr>
<tr>
<td>retargeting</td>
<td>Applying animations that were created for one model to another.</td>
</tr>
<tr>
<td>scripts</td>
<td>Used for creating logic and behaviors for your game project. You can create scripts with one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Script Canvas is a visual scripting tool that doesn't require you to know how to code. You can use the Script Canvas editor to create Script Canvas graphs. Script Canvas graphs have the file extension, .scriptcanvas, such as myscriptexample.scriptcanvas. You can add your Script Canvas graph to your game entities with the Script Canvas component.</td>
</tr>
<tr>
<td></td>
<td>• Lua is a lightweight and embeddable scripting language. You can use Lua to facilitate quick iteration of your game project. Lua scripts have the file extension .lua or .luac, such as myscriptexample.lua. You can add script functionality to your game entities with the Lua Script component.</td>
</tr>
<tr>
<td>sequence</td>
<td>The content generated from the Track View for cutscenes or other canned animation triggers.</td>
</tr>
<tr>
<td>shadow map</td>
<td>A technique for controlling how shadows are added to a scene. You can use multiple, cascaded shadow maps to control how sun shadows look at varying distances.</td>
</tr>
<tr>
<td>skinning</td>
<td>The process of binding bone joints to a model's mesh (skin).</td>
</tr>
<tr>
<td>skybox</td>
<td>A cube without the bottom side that contains the environment around a scene. Usually viewed from the inside of the cube.</td>
</tr>
<tr>
<td>slices</td>
<td>Cascaded data management system for entities. Similar to the capability of prefabs, slices are reusable component entity templates that you can easily update.</td>
</tr>
<tr>
<td>socket</td>
<td>A pivot point on a character where attachments are connected. Attachments dangle or move according to the properties of the socket.</td>
</tr>
<tr>
<td>specular map</td>
<td>An image that determines the shininess of each area of a surface.</td>
</tr>
<tr>
<td>SPOM</td>
<td>Silhouette parallax occlusion mapping. SPOM is similar to POM (p. 3284), but affects the silhouette of a mesh similar to tessellation, without the object actually being tessellated.</td>
</tr>
<tr>
<td>sprite</td>
<td>A 2D bitmap image. Multiple sprites can be grouped into a single image known as a sprite sheet.</td>
</tr>
<tr>
<td>SSDO</td>
<td>Screen Space Directional Occlusion is a method for approximating real time global illumination (GI).</td>
</tr>
<tr>
<td>SSS index</td>
<td>Subsurface scattering index. SSS is used to simulate the diffusion and scattering of light transmitted through translucent objects.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tessellation</td>
<td>The deformation of a surface using one or more geometric objects with no overlaps or gaps. Tessellation increases the geometry count of the mesh by subdividing polygons into smaller polygons before it gets displaced.</td>
</tr>
<tr>
<td>texture mapping</td>
<td>The application of an image to a surface.</td>
</tr>
<tr>
<td>TOD</td>
<td>The time of day in a level. TOD is used to simulate the changing lighting conditions as the sun crosses the sky.</td>
</tr>
<tr>
<td>UV mapping</td>
<td>The projection of texture coordinates onto a 3D surface.</td>
</tr>
<tr>
<td>vertex color</td>
<td>A method for adding variety, depth, and color variations to an object surface.</td>
</tr>
<tr>
<td>virtual reality</td>
<td>Technology that replicates the gaming environment and simulates a user's presence in it, allowing the player to feel as if they are in the game world as they interact with the environment, characters, and objects.</td>
</tr>
<tr>
<td>voxel</td>
<td>A volumetric point in a 3D space, similar to a pixel in a 2D space.</td>
</tr>
<tr>
<td>Waf</td>
<td>Game build system that allows you to automatically compile a game that targets all supported platforms.</td>
</tr>
<tr>
<td>white point</td>
<td>The reference value used to indicate true white in an image or level.</td>
</tr>
<tr>
<td>z-fighting</td>
<td>Phenomenon in 3D rendering that occurs when two or more primitives have similar or identical values in the z-buffer. Also called stitching.</td>
</tr>
</tbody>
</table>